Multipole-to-local operator in the Fast Multipole Method: comparison of FFT, rotations and BLAS improvements

Pierre Fortin

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Multipole-to-local operator in the Fast Multipole Method: comparison of FFT, rotations and BLAS improvements

Pierre Fortin

N° 5752

Novembre 2005

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--5752--FR+ENG

Multipole-to-local operator in the Fast Multipole Method: comparison of FFT, rotations and BLAS improvements

Pierre Fortin

Thème NUM — Systèmes numériques
Projet ScAlApplix

Rapport de recherche n° 5752 — Novembre 2005 — 65 pages

Abstract: In the Fast Multipole Method, most of the far field computation is due to the multipole-to-local ($M2L$) operator. In this report we distinguish two different expressions for this operator: while the first one is natural and efficient, and thus commonly used, the second one, unlike the first, respects a sharp error bound, which is proven here. Two schemes, that reduce the operation count of the $M2L$ operator, are detailed: the (block) Fast Fourier Transform and the rotations. We then present a matrix approach that uses BLAS (Basic Linear Algebra Subprograms) routines to speed up the $M2L$ computation. In order to use the more efficient level 3 BLAS (for matrix products), we require recopies, but this additional cost can be avoided thanks to special data storages. Finally all these schemes are compared, theorically and practically with uniform distributions, which validates our BLAS version.

Key-words: Fast Multipole Method, uniform distribution, error bound, Fast Fourier Transform, rotation, BLAS

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 — Télécopie : +33 1 72 92 59 ??

L'opérateur «multipole-to-local» dans la méthode multipôles rapides : comparaison des améliorations avec FFT, rotations et BLAS

Résumé : Dans la méthode multipôles rapides, la majeure partie du temps de calcul du champ lointain est due à l'opérateur «multipole-to-local» ($M2L$). Dans ce rapport, nous distinguons deux expressions différentes pour cet opérateur: alors que la première est naturelle et efficace, et donc couramment utilisée, nous prouvons ici que la second respecte, à la différence de la première, une borne d'erreur précise. Deux méthodes sont d'abord détaillées pour réduire la complexité de l'opérateur $M2L$: la Transformée Rapide de Fourier par blocs et les rotations. Nous présentons ensuite une approche différente pour accélérer le calcul de cet opérateur grâce à l'utilisation de routines BLAS (Basic Linear Algebra Subprograms). Afin d'appeler les BLAS de niveau 3, qui correspondent aux produits matriciels, et qui sont plus efficaces, nous avons recours à des recopies, mais ce surcoût peut être évité grâce à une organisation appropriée des données en mémoire. Enfin, toutes ces améliorations sont comparées, sur un plan théorique et sur un plan pratique avec des distributions uniformes. Ces comparaisons valident notre approche matricielle basée sur la bibliothèque BLAS.

Mots-clés : méthode multipôle rapide, distribution uniforme, borne d'erreur, Transformée Rapide de Fourier, rotation, BLAS

Contents

1 Introduction

1.1 FMM presentation

The N-body problem in numerical simulations describes the computation of all pairwise interactions among N bodies. The Fast Multipole Method (FMM) solves this N-body problem for any given precision with $\mathcal{O}(N)$ runtime complexity against $\mathcal{O}(N^2)$ for the direct computation. This method has been developed by Greengard & Rokhlin in [GR87] for its $2D$ uniform version, in [GR88] for its $3D$ uniform version, and in [CGR88] for its adaptive version. First introduced for gravitational potentials in astrodynamics or electrostatic (coulombic) potentials in molecular dynamics [GR87], it has then been extended with different mathematical bases to electromagnetism [Dar99], VLSI capacitance [NW91], radiosity [SHT+95], object modeling [CBC+01] and many more: we will consider here gravitational and electrostatic potentials.

For these potentials the $3D$ FMM has the drawback to present a linear complexity with an underlying factor in $\mathcal{O}(P^4)$ where P is the maximum degree in the expansions. In contrary the $2D$ FMM constant is in $\mathcal{O}(P^2)$. As this limits the use of FMM in $3D$ simulations, some schemes have been introduced in order to reduce this cost such as: Fast Fourier Transform (FFT) in [EB96], rotations in [WHG96] and more recently plane wave expansions in [GR97] and in [CGR99]. All these schemes reduce the $\mathcal{O}(P^4)$ factor to a $\mathcal{O}(P^3)$ or $\mathcal{O}(P^2)$.

We propose here a different approach: we will use highly efficient implementation techniques such as BLAS (Basic Linear Algebra Subprograms) [LHKK79] [DCHH88] [DCHD90] to improve the runtime of FMM. Thanks to optimal use of the different layers of the hierarchical memory of the computer, so that the pipelines of the floating point units are filled at best, they indeed offer substantial runtime speedup on superscalar architectures. This speedup only affects the constant in the $\mathcal{O}(P^4)$ notation and we keep the $\mathcal{O}(P^4)$ operation count but since, in molecular dynamic simulations for example, P usually ranges from 3 to 15, which is quite low in terms of operation count, the speedup obtained with BLAS can exceed the one obtained with a lower operation count.

We also distinguish two different expressions of the $M2L$ operator. Indeed, the error bound of the 3D FMM has been historically ([GR87] [Gre88]) presented for the evaluation of potential with either finite multipole expansions or finite local expansions. But, as mentioned by several authors ([SP97], [WHG94]), we have also to consider, when implementing the FMM, that the $M2L$ operator acts on finite multipole expansions, which means that both multipole and local expansions are finite. When denoting P the maximum degree of the expansions, and n (respectively j) the degrees of the multipole (respectively local) expansion terms, two different kinds of $M2L$ expressions can then be used. In the first one, both n and j fully range between 0 and P, whereas in the second $M2L$ expression we use only terms with: $n+j \leq P$. While the first one is natural and commonly used ([PSS95], [SP97], [WHG94]), no sharp error bound has yet been found, to our knowledge, for the corresponding summations. We will prove that the second one, though generally less efficient, respects such sharp error bound. For these two expressions, we will present the principles and the implementation features for two improvements of the $M2L$ operator, namely FFT and rotations, as well as for our BLAS approach. We will then propose a detailed comparaison of the memory requirements, numerical precisions and runtimes, for uniform distributions sequentially computed, between these three schemes depending on the $M2L$ expression used.

Before that, we here briefly describe the FMM: we refer the reader to the articles of Greengard & Rokhlin for a complete presentation of the FMM and a definition of the following terms. The FMM uses a quadtree or an octree in order to divide the computational space. The potential field in each point is then decomposed in a near field and a far field part: the near field is directly computed whereas the far field is approximated thanks to multipole and local expansions. The multipole expansions are built at the leaf level and we deduce the local expansions in each leaf after an upward and a downward pass. During the downward pass all multipole expansions in the *interaction list* of a given cell c are converted to local expansions for c: this operation is valid since c is *well-separated* from each member of its interaction list. We here use the same definition as Greengard & Rokhlin [GR97] for the notion of *well-separateness*: two cells are well-separated if they do not share a boundary point. More precisely, when denoting ws the well-separateness criterion, with $ws = 1$ only the nearest neighbors are considered as not well-separated while with $ws = 2$ both nearest neighbors and second nearest neighbors are considered as not well-separated. We use $ws = 1$ in this paper which implies 189 members in each interaction list.

We denote by $P2M$ the operator that determines the multipole expansion of a leaf due to the particles it contains. $M2M$ denotes the translation of a multipole expansion, $M2L$ the conversion of a multipole expansion into a local expansion, and $L2L$ the translation of a local expansion. We consider the root of the octree (or

computational box) to be at level 0. The *height* of the octree is defined as its maximum level: an octree with only one root have a height of 0, just like an empty octree.

We focus in this paper on uniform distributions sequentially computed. The uniform octree has been mainly implemented as in the $DPMTA$ (Distributed Parallel Multipole Tree Algorithm) code (see [Ell95] or [Ran99]): we use Morton ordering for the indexing of cells and, given a cell index, all indexes of the interaction list of this cell can be quickly computed thanks to bit operations. This is also valid for accesses to the father of a cell, to its children and to the cells located in the near field of a leaf. Moreover as in $DPMTA$ all $M2L$ *transfer functions* (see section 1.2.1) are precomputed at each level during the downward pass: when we need to perform one $M2L$ operation the corresponding $M2L$ transfer function is retrieved from an adequate data structure thanks to the corresponding $M2L$ vector.

The rest of this paper is organised as follows: in the rest of this section we will introduce the formulae used in our implementation of the FMM. We will also discuss one error bound issue and present the two different expressions of the $M2L$ operator. We will expose and discuss the FFT enhancement in section 2, the use of rotations in section 3, and the introduction of BLAS in section 4. All these improvements have been implemented in a code named FMB for Fast Multipoles with BLAS (or Fast Multipole in Bordeaux) thus allowing precise comparisons among them as presented in section 5.

1.2 The error bound in the Fast Multipole Method

We introduce here the formulae used in our implementation of the Fast Multipole Method. We especially focus on the conversion of a multipole expansion to a local expansion ($M2L$ operator), its two different expressions and their implications on the error bound of the method.

1.2.1 Formulae for the Fast Multipole Method

The original formulae of the Fast Multipole Method for the evaluation of Coulombic or gravitational interactions of particles with an uniform distribution in a $3D$ space were first given by Greengard & Rokhlin in [GR88].

However since the $M2L$ operations are the most time-consuming part of the far field computation, modifications of the original formulae have been introduced in order to simplify the $M2L$ operator: we can here cite Epton & Dembart [ED95], White & Head-Gordon [WHG94], and the works done at Duke University (see for example [Ell95]). One important requirement in the choice of these formulae is the observance of the symmmetry property among the multipole and local expansion terms of opposite order (see equation (A.3) in appendix A): hence only terms with positive orders need to be computed.

We have choosen the formulae of Epton & Dembart, mainly because of the clarity of the underlying theorems and their proofs.

FMM operators. With the associated Legendre functions P_l^m and the spherical harmonics Y_l^m defined in appendix A, we can describe the operators that handle the multipole and local expansions. Like Epton & Dembart [ED95], we first respectively define the *Outer* and *Inner* functions by:

$$O_l^m(r,\theta,\phi) = \frac{(-1)^l i^{|m|}}{A_l^m} Y_l^m(\theta,\phi)\frac{1}{r^{l+1}} \qquad \forall\ (l,m) \in \mathbb{N}^2 \quad with \quad 0 \leq |m| \leq l, \tag{1.1}$$

and:

$$I_l^m(r,\theta,\phi) = i^{-|m|} A_l^m Y_l^m(\theta,\phi) r^l \qquad \forall\ (l,m) \in \mathbb{N}^2 \quad with \quad 0 \leq |m| \leq l, \tag{1.2}$$

where:

$$A_l^m = \frac{(-1)^l}{\sqrt{(l-m)!(l+m)!}}.$$

As for the spherical harmonics, we emphasize that the O_j^k and I_j^k functions are considered as identically null for $j < 0$ or $|k| > j$. They can also be written as:

$$O_l^m(r,\theta,\phi) = i^{-|m|}(l-|m|)! P_l^{|m|}(\cos\theta) e^{i.m.\phi} \frac{1}{r^{l+1}}, \tag{1.3}$$

$$I_l^m(r,\theta,\phi) = \frac{(-1)^l i^{|m|}}{(l+|m|)!} P_l^{|m|}(\cos\theta) e^{i.m.\phi} r^l. \tag{1.4}$$

And the relations among opposite orders are:

$$O_l^{-m} = (-1)^m \overline{O}_l^m, \tag{1.5}$$

$$I_l^{-m} = (-1)^m \overline{I}_l^m. \tag{1.6}$$

Letting $\mathbf{X}$ and $\mathbf{X}'$ be two position vectors in $3D$ space, we now give the main theorems that the FMM requires: their proof can be found in [ED95].

Theorem 1 (Classical translation theorem). *Under the assumption $\|\mathbf{X}\| > \|\mathbf{X}'\|$, we have:*

$$\frac{1}{\|\mathbf{X} - \mathbf{X}'\|} = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} (-1)^n I_n^{-l}(\mathbf{X}') O_n^l(\mathbf{X}).$$

This formula is obtained by a decomposition of the potential with Legendre polynomials which are themselves decomposed thanks to the Addition Theorem for spherical harmonics.

The next theorem is used to establish $M2M$ (*Outer-to-Outer*) and $M2L$ (*Outer-to-Inner*) operators.

Theorem 2 (Outer-to-Outer, Outer-to-Inner Laplace translation theorem). *Let $\|\mathbf{X}\| > \|\mathbf{X}'\|$, then:*

$$O_n^l(\mathbf{X} - \mathbf{X}') = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} (-1)^j I_j^{-k}(\mathbf{X}') O_{n+j}^{l+k}(\mathbf{X}) \qquad \forall\ (n,l) \in \mathbb{N}^2 \quad with \quad 0 \leq |l| \leq n.$$

The last theorem corresponds to the *Third Addition Theorem* in [Gre88], and it is used for $L2L$ (*Inner-to-Inner*) operator:

Theorem 3 (Inner-to-Inner Laplace translation theorem).

$$I_n^l(\mathbf{X} - \mathbf{X}') = \sum_{j=0}^{n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{X}') I_{n-j}^{l-k}(\mathbf{X}) \qquad \forall\ (n,l) \in \mathbb{N}^2 \quad with \quad 0 \leq |l| \leq n.$$

We now give the operators $P2M$, $M2M$, $M2L$ and $L2L$ that are needed for the FMM, as expressed in [ED95]. We also give the formulae that are used to deduce the potential and the forces from the local expansions.

Definition 1.1 ($P2M$ operator). *Given m charges $(q_i)_{i \in [\![1,m]\!]}$ located in $\mathbf{Q_i} = (\rho_i, \alpha_i, \beta_i)$ whose relative coordinates according to a center $\mathbf{z_0}$ are: $\mathbf{Q_i} - \mathbf{z_0} = (\rho_i', \alpha_i', \beta_i')$, we define:*

$$M_j^k = (-1)^j \sum_{i=1}^{m} q_i I_j^k(\rho_i', \alpha_i', \beta_i') \qquad \forall\ (j,k) \in \mathbb{N}^2 \quad with \quad 0 \leq |k| \leq j.$$

The potential is then given by the following multipole expansion:

$$\Phi(\mathbf{x}) = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} M_j^k O_j^{-k}(\mathbf{x} - \mathbf{z_0}).$$

M_j^k are denoted as *multipole expansion terms*. The symmetry property (1.6) among opposite orders is also valid for these multipole expansion terms:

$$M_j^{-k} = (-1)^k \overline{M}_j^k. \tag{1.7}$$

Definition 1.2 ($M2M$ operator). *${M'}_n^l$ (with $n \geq 0$, $|l| \leq n$) being the former multipole expansion terms, whose center is $\mathbf{z_0}$, we have for the new multipole expansion terms, whose center is $\mathbf{z_1}$:*

$$M_j^k = \sum_{n=0}^{j} \sum_{\substack{l=-n, \\ |k-l| \leq j-n}}^{n} {M'}_n^l I_{j-n}^{k-l}(\rho, \alpha, \beta) \qquad \forall\ (j,k) \in \mathbb{N}^2 \quad with \quad 0 \leq |k| \leq j$$

where (ρ, α, β) are the spherical coordinates of the vector: $\mathbf{z_1} - \mathbf{z_0}$.

Definition 1.3 ($M2L$ operator). *M_n^l (with $n \geq 0$, $|l| \leq n$) being the multipole expansion terms, whose center is $\mathbf{z_1}$, we have for the local expansion terms, whose center is $\mathbf{z_2}$:*

$$L_j^k = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho, \alpha, \beta) \qquad \forall\ (j,k) \in \mathbb{N}^2 \quad with \quad 0 \leq |k| \leq j$$

where (ρ, α, β) are the spherical coordinates of the $M2L$ vector: $\mathbf{z_2} - \mathbf{z_1}$.

The potential is now given by the following local expansion:

$$\Phi(\mathbf{x}) = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} L_j^k I_j^k(\mathbf{x} - \mathbf{z_2}). \tag{1.8}$$

L_j^k are denoted as *local expansion terms*. The symmetry property (1.6) among opposite orders is also valid for these local expansion terms:

$$L_j^{-k} = (-1)^k \overline{L}_j^k. \tag{1.9}$$

This can be proved by considering that $\Phi(\mathbf{x}) \in \mathbb{R}$ and that the L_j^k are unique. Therefore only the terms with positive orders are computed when implementing $M2L$, which save half of the computation.

Definition 1.4 ($L2L$ operator). *L'^l_n (with $n \geq 0$, $|l| \leq n$) being the former local expansion terms, whose center is $\mathbf{z_2}$, we have for the new local expansion terms, whose center is $\mathbf{z_3}$:*

$$L_j^k = \sum_{n=j}^{+\infty} \sum_{\substack{l=-n, \\ |l-k| \leq n-j}}^{n} L'^l_n I_{n-j}^{l-k}(\rho, \alpha, \beta) \qquad \forall\ (j,k) \in \mathbb{N}^2 \quad with \quad 0 \leq |k| \leq j$$

where (ρ, α, β) are the spherical coordinates of the vector: $\mathbf{z_3} - \mathbf{z_2}$. When dealing with finite expansions of maximum degree P, the local expansion terms of degree strictly greater than P are null and the infinite sum is then written as: $\sum_{n=j}^{P}$.

With $M2M$, $M2L$ and $L2L$ operators we always converte or translate an old (multipole or local) expansion to a new (multipole or local) expansion thanks to some O_j^k or some I_j^k. These latters are named $M2M$, $M2L$ or $L2L$ *transfer functions*.

Evaluation of potential and forces. We first consider the evaluation of the potential. For a finite local expansion with maximum degree P given by L_j^k, $0 \leq j \leq P$, $|k| \leq j$, we can evaluate the potential at a point $\mathbf{Z}$ with (1.8):

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(r, \theta, \phi). \tag{1.10}$$

However due to the properties (1.6) and (1.9), we have: $L_j^k I_j^k + L_j^{-k} I_j^{-k} = L_j^k I_j^k + \overline{L}_j^k \overline{I}_j^k = 2\ Re(L_j^k I_j^k)$, where Re denotes the real part of a complex number. The potential may thus be evaluated in this faster way:

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{P} \left[L_j^0 I_j^0(r, \theta, \phi) + \sum_{k=1}^{j} 2\ Re(L_j^k I_j^k(r, \theta, \phi)) \right].$$

For the evaluation of the force, we start with equation (1.10), and we want to compute:

$$\mathbf{F}(\mathbf{Z}) = -\mathbf{grad}\ \Phi(\mathbf{Z}),$$

that is to say, in the local base of the spherical coordinates:

$$F_r(\mathbf{Z}) = -\frac{\partial \Phi(\mathbf{Z})}{\partial r}, \quad F_\theta(\mathbf{Z}) = -\frac{1}{r} \frac{\partial \Phi(\mathbf{Z})}{\partial \theta}, \quad F_\phi(\mathbf{Z}) = -\frac{1}{r \sin\theta} \frac{\partial \Phi(\mathbf{P})}{\partial \phi}.$$

The gradient in spherical coordinates (with our agreement of θ being the colatitudinal coordinate and ϕ being the longitudinal coordinate) can be expressed as:

$$\mathbf{grad} f = \frac{\partial f}{\partial r} \mathbf{e_r} + \frac{1}{r} \frac{\partial f}{\partial \theta} \mathbf{e}_\theta + \frac{1}{r \sin\theta} \frac{\partial f}{\partial \phi} \mathbf{e}_\phi$$

where $(\mathbf{e_r}, \mathbf{e}_\theta, \mathbf{e}_\phi)$ is the local base of the spherical coordinates. This formula is due to the fact that the elementary move in spherical coordinates is: $\mathbf{dl} = dr\mathbf{e_r} + rd\theta\mathbf{e}_\theta + r\sin\theta d\phi\mathbf{e}_\phi$. Since:

$$\frac{\partial I_j^k(r,\theta,\phi)}{\partial r} = \frac{j}{r} I_j^k(r,\theta,\phi), \quad \frac{\partial I_j^k(r,\theta,\phi)}{\partial \theta} = i^{-|k|} A_j^k r^j \frac{\partial Y_j^k(\theta,\phi)}{\partial \theta}, \quad \frac{\partial I_j^k(r,\theta,\phi)}{\partial \phi} = ikI_j^k(r,\theta,\phi),$$

we have, in the local base of the spherical coordinates:

$$\begin{aligned} F_r(\mathbf{Z}) &= -\sum_{j=1}^{P}\sum_{k=-j}^{j} \frac{j}{r} L_j^k I_j^k(r,\theta,\phi) \\ &= -\frac{1}{r}\left(\sum_{j=1}^{P} jL_j^0 I_j^0(r,\theta,\phi) + \sum_{k=1}^{j} 2j\ Re\left(L_j^k I_j^k(r,\theta,\phi)\right)\right) \end{aligned}$$

$$\begin{aligned} F_\theta(\mathbf{Z}) &= -\frac{1}{r}\sum_{j=0}^{P}\sum_{k=-j}^{j} L_j^k \frac{\partial I_j^k(\theta,\phi)}{\partial \theta} \\ &= -\frac{1}{r}\left(\sum_{j=0}^{P} L_j^0 \frac{\partial I_j^0(r,\theta,\phi)}{\partial \theta} + \sum_{k=1}^{j} 2\ Re\left(L_j^k \frac{\partial I_j^k(r,\theta,\phi)}{\partial \theta}\right)\right) \end{aligned}$$

$$\begin{aligned} F_\phi(\mathbf{Z}) &= -\frac{1}{r\sin\theta}\sum_{j=0}^{P}\sum_{k=-j}^{j} ikL_j^k I_j^k(r,\theta,\phi) \\ &= -\frac{1}{r\sin\theta}\sum_{j=0}^{P}\sum_{k=1}^{j}(-2k)\ Im\left(L_j^k I_j^k(r,\theta,\phi)\right) \end{aligned}$$

where *Im* denotes the imaginary part of a complex number.

Remark 1.1. *The* $\frac{\partial Y_j^k(\theta,\phi)}{\partial\theta}$ *are computed thanks to the following equation:* $\frac{dP_l^m(\cos\theta)}{d\theta} = \frac{l\cos\theta P_l^m(\cos\theta) - (l+m)P_{l-1}^m(\cos\theta)}{\sin\theta}$, $\forall\ l \geq 1$, $\forall\ |m| \leq l-1$. *And for* $m = l$ *:* $\frac{dP_l^l(\cos\theta)}{d\theta} = \frac{l\cos\theta P_l^l(\cos\theta)}{\sin\theta}$. *This can be proved with:* $P_l^{m+1}(z) = (z^2-1)^{-\frac{1}{2}}\left((l-m)\,zP_l^m(z) - (l+m)\,P_{l-1}^m(z)\right)$ *(see [AS72], chapter 8 about Legendre Functions, page 333).*

1.2.2 Error bound analysis

As exposed in the introduction, the $M2L$ operator concretely uses finite multipole expansion to compute (finite) local expansions. When denoting P as the maximum degree of the expansions, whose terms have degrees therefore ranging from 0 to P, two different kinds of $M2L$ expressions can then be used.

The first and the most common $M2L$ expression is:

$$L_j^k = \sum_{n=0}^{P}\sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho,\alpha,\beta).$$

We use here all the multipole expansion terms M_n^l (which vanish outside the range $n \geq 0$, $|l| \leq n$) and we also use O_j^k terms with degrees up to $2P$. Even if some interesting works have been done to estimate the behavior of the error induced by such $M2L$ expression (see [PSS95] [SP97]), no sharp error bound has been found yet. We name this $M2L$ expression *M2L kernel*[1] *with double height* or *UpTo2P*.

A second $M2L$ expression is:

$$L_j^k = \sum_{n=0}^{P-j}\sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho,\alpha,\beta).$$

We here limit the maximum degree of the O_j^k used to P: we thus name this $M2L$ expression *M2L kernel with single height* or *UpToP*. This has been used for example in the DPMTA code developped at Duke University but the proof in the error bound given in [Ell95] is wrong[2]. We however recommend the reading of this appendix

[1] The *kernel* name has been inspired by [Ell95].

[2] The equation (C.23) in appendix C of [Ell95] is wrong because the composition of differential operators differs from the product of the derivatives.

since it presents a worst case error analysis similar to the one used hereafter. In particular it shows that no additional error is introduced by the $M2M$ operation. As explained in [Gre88] the $L2L$ operation does not introduce any error at all. We can thus focus on the $M2L$ operation.

In the following we prove that an $M2L$ operator with single height kernel respects a sharp error bound (similar to the one presented in [Ell95]). It has to be noted that this error bound for single height $M2L$ kernel was briefly presented by [WHG94], but they have used double height kernel in their implementation.

In order to prove this error bound, we study the potential in point $\mathbf{Z}$ due to one single unit charge located in $\mathbf{Q}$ as pictured in figure 1.1. We denote by $\mathcal{B}_1$ (respectively $\mathcal{B}_2$) the ball centered in $\mathbf{X_1}$ with radius r_1

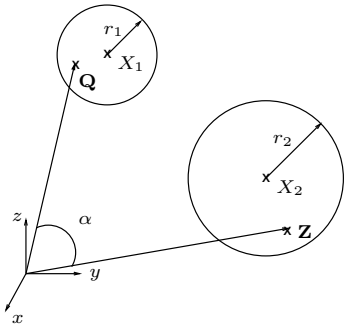


Figure 1.1: Our problem configuration.

(respectively X_2 and r_2). $\mathbf{Z}$ is enclosed in $\mathcal{B}_1$, $\mathbf{Q}$ in $\mathcal{B}_2$. Moreover, denoting $R = \|\mathbf{X_1} - \mathbf{X_2}\|$, we assume that: $R > r_1 + r_2$.

Denoting α the angle between "vectors" $\mathbf{Q}$ and $\mathbf{Z}$, $r_>$ (respectively $r_<$) the maximum (respectively minimum) between the norm of $\mathbf{Q}$ and the one of $\mathbf{Z}$, and P_n the Legendre polynomials, we have:

$$\Phi(\mathbf{Z}) = \frac{1}{\|\mathbf{Z} - \mathbf{Q}\|} = \sum_{n=0}^{+\infty} \frac{r_<^n}{r_>^{n+1}} P_n(\cos\alpha). \tag{1.11}$$

Defining $\Phi_P(\mathbf{Z})$ the potential with finite summation up to P, we have:

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| = \sum_{n=P+1}^{+\infty} \frac{r_<^n}{r_>^{n+1}} P_n(\cos\alpha).$$

As for all $n \geq 0$ and for all $x \in [-1, 1]$, we have $|P_n(x)| \leq 1$, we can write:

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \sum_{n=P+1}^{+\infty} \frac{r_<^n}{r_>^{n+1}} = \frac{1}{r_> - r_<} \left(\frac{r_<}{r_>}\right)^{P+1}. \tag{1.12}$$

This leads to the following error bound:

Proposition 1.1. *$\forall\ \mathbf{Q} \in \mathcal{B}_1$, and $\forall\ \mathbf{Z} \in \mathcal{B}_2$, we have, under the condition $R > r_1 + r_2$:*

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{r_1 + r_2}{R}\right)^{P+1}.$$

Proof. Since: $\|\mathbf{Z} - \mathbf{Q}\| = \|\left((\mathbf{Z} - \mathbf{X_2}) + (\mathbf{X_1} - \mathbf{Q})\right) + (\mathbf{X_2} - \mathbf{X_1})\|$, this results directly from the inequality (1.12).

We now decompose the potential in the same way the FMM does while maintaining this error bound, and prove the following theorem.

Theorem 4. *The $M2L$ operator* **with single height** $M2L$ **kernel** *strictly respects the error bound in proposition 1.1.*

Proof. In order to prove theorem 4, we consider a multipole expansion, centered in $\mathbf{X_1}$, and a local expansion, centered in $\mathbf{X_2}$, to express the potential at $\mathbf{Z}$ due to the particle in $\mathbf{Q}$. We start with:

$$\Phi(\mathbf{Z}) = \frac{1}{\|\mathbf{Z}-\mathbf{Q}\|} = \frac{1}{\|\left(\mathbf{X_2}-\mathbf{X_1}\right) - \left((\mathbf{Q}-\mathbf{X_1}) - (\mathbf{Z}-\mathbf{X_2})\right)\|}.$$

Since the condition $R > r_1 + r_2$, which guarantees the convergence of both multipole and local expansions, implies $\|(\mathbf{Q}-\mathbf{X_1}) - (\mathbf{Z}-\mathbf{X_2})\| < \|\mathbf{X_2}-\mathbf{X_1}\|$, we can write with the classical translation theorem (theorem 1):

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty}\sum_{l=-n}^{n} (-1)^n I_n^{-l}\left((\mathbf{Q}-\mathbf{X_1}) - (\mathbf{Z}-\mathbf{X_2})\right) O_n^l\left(\mathbf{X_2}-\mathbf{X_1}\right).$$

Since $I_n^l(-\mathbf{X}) = (-1)^n I_n^l(\mathbf{X})$, we have:

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty}\sum_{l=-n}^{n} I_n^{-l}((\mathbf{Z}-\mathbf{X_2}) - (\mathbf{Q}-\mathbf{X_1}))O_n^l(\mathbf{X_2}-\mathbf{X_1}).$$

We then use the Inner-to-Inner translation theorem (theorem 3), which does not require any condition on $\mathbf{X}$ and $\mathbf{X}'$, and obtain:

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty}\sum_{l=-n}^{n}\sum_{j=0}^{n}\sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z}-\mathbf{X_2}) O_n^l(\mathbf{X_2}-\mathbf{X_1}). \tag{1.13}$$

We emphasize here that the equation (1.13) is just a rewriting of equation (1.11): that's why when troncating the series in equation (1.13) for $n > P$ we obtain the same error bound as in proposition (1.1). Thus we have:

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P}\sum_{l=-n}^{n}\sum_{j=0}^{n}\sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z}-\mathbf{X_2}) O_n^l(\mathbf{X_2}-\mathbf{X_1}).$$

We can inverse the two finite summations on the degrees n and j like:

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P}\sum_{k=-j}^{j}\sum_{n=j}^{P}\sum_{l=-n}^{n} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z}-\mathbf{X_2}) O_n^l(\mathbf{X_2}-\mathbf{X_1}).$$

We first introduce $n' = n - j$:

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P}\sum_{k=-j}^{j}\sum_{n'=0}^{P-j}\sum_{l=-(n'+j)}^{n'+j} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n'}^{-l-k}(\mathbf{Z}-\mathbf{X_2}) O_{n'+j}^l(\mathbf{X_2}-\mathbf{X_1}),$$

and $l' = l + k$ to obtain:

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P}\sum_{k=-j}^{j}\sum_{n'=0}^{P-j}\sum_{l'=-(n'+j)+k}^{n'+j+k} (-1)^j I_j^k(\mathbf{Q}-\mathbf{X_1}) I_{n'}^{-l'}(\mathbf{Z}-\mathbf{X_2}) O_{n'+j}^{l'-k}(\mathbf{X_2}-\mathbf{X_1}).$$

We remember here that $I_{n'}^{-l'}(\mathbf{x})$ imposes: $-n' \leq l' \leq n'$. In the same way, $I_j^k(\mathbf{x})$ imposes: $|k| \leq j$, that is to say: $j + k \geq 0$, and: $k - j \leq 0$. Moreover:

$$\begin{array}{lll} j+k \geq 0 & \Rightarrow & n'+j+k \geq n', \\ k-j \leq 0 & \Rightarrow & -(n'+j)+k = -n'+(k-j) \leq -n'. \end{array}$$

Under these conditions we have the following equality:

$$\sum_{l'=-(n'+j)+k}^{n'+j+k} = \sum_{l'=-n'}^{n'}.$$

Which means that the set of the terms which are indexed by the summation on the right side is included in the set of terms which are indexed by the summation on the left side, and all the terms present on the left side, but not on the right side, vanish.

With $l = -l'$ and $n = n'$, we finally have:

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} \sum_{n=0}^{P-j} \sum_{l=-n}^{n} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_n^l(\mathbf{Z} - \mathbf{X_2}) O_{n+j}^{-l-k}(\mathbf{X_2} - \mathbf{X_1}).$$

Let's remark that:

$$\sum_{j=0}^{P} \sum_{n=0}^{P-j} \equiv \sum_{\substack{|j| \leq P \\ |n| \leq P, \\ 0 \leq n+j \leq P}} \equiv \sum_{n=0}^{P} \sum_{j=0}^{P-n}.$$

We obtain:

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} \left(\sum_{j=0}^{P-n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) O_{n+j}^{-l-k}(\mathbf{X_2} - \mathbf{X_1}) \right) I_n^l(\mathbf{Z} - \mathbf{X_2}).$$

For all $\mathbf{Q} \in \mathcal{B}_1$, the multipole expansions terms M_j^k, $\forall\ (j,k) \in \mathbb{N}^2$ with $0 \leq |k| \leq j$, being defined by: $M_j^k = (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1})$, we have thus proved that:

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} L_n^l I_n^l(\mathbf{Z} - \mathbf{X_2}),$$

where the L_n^l denote the local expansion terms, centered in $\mathbf{X_2}$, due to the unit charge located in $\mathbf{Q}$, and obtained thanks to the $M2L$ operator with single height $M2L$ kernel applied to the multipole expansion terms M_j^k as:

$$L_n^l = \sum_{j=0}^{P-n} \sum_{k=-j}^{j} M_j^k O_{n+j}^{-l-k}(\mathbf{X_2} - \mathbf{X_1})$$

□

It has to be noted that the condition of well-separateness, with either $ws = 1$ or $ws = 2$, is in fact more strict than $R > r_1 + r_2$ in order to ensure a fast enough convergence. With $ws = 1$ for example, we have, for a cell of side l: $r_1 = r_2 = \frac{\sqrt{3}}{2}$ and $R \geq 2$. Thus:

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{\sqrt{3}}{2} \right)^{P+1} = \frac{1}{R - (r_1 + r_2)} (0.866)^{P+1}.$$

With $ws = 2$, $(R \geq 3)$, we obtain:

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{\sqrt{3}}{3} \right)^{P+1} = \frac{1}{R - (r_1 + r_2)} (0.577)^{P+1}.$$

This error bound is worst than the one presented in [Gre88] which looks like $\left(\frac{r_2}{R - r_1}\right)^{P+1}$ ($ws = 1 \Rightarrow (0.764)^{P+1}$ and $ws = 2 \Rightarrow (0.406)^{P+1}$).

As already remarked (see [GR97]) the error due to $M2L$ is also higher than the error due to the simple evaluation of the potential with multipole expansions which looks like $\left(\frac{r_1}{R'}\right)^{P+1}$ where R' denotes the minimal distance between the center of the multipole expansion and the point where the potential is computed ($ws = 1 \Rightarrow (0.577)^{P+1}$ and $ws = 2 \Rightarrow (0.346)^{P+1}$).

In conclusion, since the error in proposition (1.1) is higher than the one due to the multipole expansion terms (see [GR97]), and since no additional error is introduced by the $M2M$ and $L2L$ operators, the error bound of the FMM with a single height kernel $M2L$ operator is therefore the same as in proposition (1.1).

As a result, this error bound can safely be used in the case of single height kernel in order to determine the value of P according to the required precision. Moreover since the double height kernel, compared to the single height kernel, only adds terms in the $M2L$ formula, it respects at least this error bound but even adds precision with additional cost at runtime. The problem is that we cannot predict this gain in the accuracy of the FMM, and therefore we cannot compare single and double height kernels according to the tradeoff between theoretical accuracy and runtime. Only comparisons with practical accuracies will be possible as described in section 5: see also [Ell95] (section C.3.1) for a brief comparison using theoretical operation count vs. accuracy.

1.2.3 Memory requirements

For comparison purpose with the other methods presented below, we detail here the memory requirements for the original computation scheme of $M2L$, also named *classic* $M2L$. We here focus only on the memory used by the expansions. We denote by c the size of a complex number, in double precision we have: $c = 16$ bytes. As we only use in the expansions terms with positive or null orders, the size of a multipole ($\mathcal{M}$) or local ($\mathcal{L}$) expansion according to P is (see definitions 1.1 and 1.3):

$$\mathcal{M}(P) = \mathcal{L}(P) = \sum_{j=0}^{P}\sum_{k=0}^{j} c = \frac{(P+1)(P+2)}{2}c.$$

The number of cells $\mathcal{N}$ in an octree of height H is:

$$\mathcal{N}(H) = \sum_{l=0}^{H} 8^l = \frac{8^{H+1}-1}{7}.$$

We need now to measure the size of the $M2L$ transfer functions which are precomputed for each level. Their number corresponds to the number of all possible $M2L$ vectors, that is to say $\mathcal{N}_{\mathcal{T}} = (2(2ws+1)+1)^3-(2ws+1)^3$, and thus with $ws = 1$:

$$\mathcal{N}_{\mathcal{T}} = 316.$$

With single height $M2L$ kernel the size of one $M2L$ transfer function $\mathcal{T}$ equals $\mathcal{M}$, but with double height kernel, we have: $\mathcal{T}(P) = \mathcal{M}(2P)$. Therefore, the total memory requirements, *Mem*, for the classic $M2L$ computation is:

$$Mem(P,H) = \mathcal{N}(H).(\mathcal{M}(P)+\mathcal{L}(P)) + \mathcal{N}_{\mathcal{T}}.\mathcal{T}(P).$$

For single height kernel this gives:

$$Mem_{Sg}(P,H) = \mathcal{N}(H)(P+1)(P+2)c + \mathcal{N}_{\mathcal{T}}\frac{(P+1)(P+2)}{2}c, \tag{1.14}$$

and for double height kernel:

$$Mem_{Db}(P,H) = \mathcal{N}(H)(P+1)(P+2)c + \mathcal{N}_{\mathcal{T}}\frac{(2P+1)(2P+2)}{2}c. \tag{1.15}$$

1.2.4 $M2L$ operator complexity

As a reference for coming comparisons we also present the difference of complexity between single and double heights $M2L$ kernel. We recall that only local expansion terms with positive or null orders are computed. For single height, we have thus:

$$M2L = \frac{1}{12}(P+3)(P+1)(P+2)^2,$$

while for double height this is:

$$M2L = \frac{1}{2}(P+2)(P+1)^3.$$

Let's remember for the moment that the ratio among the two heights is roughly 6.

2 Fast Fourier Transform

The use of Fast Fourier Transform (FFT) in order to speed up the computation of FMM operators ($M2M$, $M2L$ and $L2L$) has been developped at Duke University by William D. Elliott ([EB96] or [Ell95]). This has been included in a code named DPMTA.

This work was performed only for single height $M2L$ kernel and the block version used to prevent numerical instability implemented in DPMTA has restricted the possible values of P to multiples of the block size.

In this section we generalize this FFT enhancement to both single and double height $M2L$ kernel and the block version presented here can be used for any P.

The FFT has been applied only to $M2L$ operator since this is the most time-consuming operator in the FMM, but it should be straightforward to derive FFT for $M2M$ (see [ED95]) and $L2L$. However, as explained in [Ell95], the full computation with the 3 consecutive operators cannot be directly performed in Fourier space.

The principle behind the use of FFT in FMM is to view $M2M$, $M2L$ and $L2L$ operators as convolution (or correlation) between 2 periodic sequences. This convolution will take place in *real* space (also named *coefficient* space by Elliott). After having been processed by a (forward) discrete Fourier transform, the 2 sequences are considered to be in *Fourier* space. The convolution in real space corresponds to a point-wise product in Fourier space, and a backward discrete Fourier transform gives the same result (thanks to the periodicity property) in real space than the original convolution.

The theory of Discrete Fourier Transform is detailed in appendix B: we will now try to apply it to $M2L$ computation.

2.1 Discrete Fourier Transform theory applied to $M2L$ operator

We first consider the $M2L$ operator with single height kernel as defined in (1.2.2). The degree of the expansion terms ranges from 0 to P. We therefore consider:

$$L_j^k = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l O_{n+j}^{-(k+l)} \qquad \forall (j,k) \in \mathbb{N}^2, j > 0, |k| < j. \tag{2.1}$$

2.1.1 $M2L$ operator viewed as a $2D$ convolution

We will rewrite the $M2L$ operator as a $2D$ convolution. We could have choosen to rewrite it as a $2D$ correlation but this would have led to an "inverse" point-wise product in Fourier space. We have preferred an inversion in real space such as, for all $(j,k) \in \mathbb{N}^2$:

$$\begin{array}{l} {L'}_j^k = L_{-j}^{-k}, \\ {O'}_j^k = O_{-j}^k \quad \text{(only degrees are inversed).} \end{array} \tag{2.2}$$

When inserted in $M2L$, we have:

$${L'}_{-j}^{-k} = \sum_{n \in \mathcal{J}} \sum_{l \in \mathcal{K}} M_n^l {O'}_{-(n+j)}^{-(k+l)} \qquad \forall (j,k) \in \mathbb{N}^2,$$

and with $j' = -j$ and $k' = -k$:

$${L'}_{j'}^{k'} = \sum_{n \in \mathcal{J}} \sum_{l \in \mathcal{K}} M_n^l {O'}_{j'-n}^{k'-l}.$$

For simplicity, we redefine hereafter the L_j^k and the O_j^k so that they correspond respectively to ${L'}_j^k$ and ${O'}_j^k$ and we obtain the following $2D$ convolution:

$$L_j^k = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l O_{j-n}^{k-l} \qquad \forall (j,k) \in \mathbb{N}^2, j > 0, |k| < j.$$

let: $\Gamma = \{(n,l) \in \mathbb{N}^2 \quad with \quad 0 \le n \le P, |l| \le n\}$, describe the set of orders and degrees for which M_j^k, O_j^k and L_j^k are defined. These terms are considered to vanish out of Γ.

We define two sets $\mathcal{J}$ and $\mathcal{K}$, as in section (B.6), by:

$$\mathcal{J} = [\![-P, P]\!],$$

$$\mathcal{K} = [\![-P, P]\!],$$

and we extend the definition of X_j^k in the following way:

$$\widetilde{X}_j^k = \begin{cases} X_j^k & \text{if } (j,k) \in \Gamma \\ 0 & \text{if } (j,k) \in \mathcal{J} \times \mathcal{K} \setminus \Gamma \end{cases} \qquad \forall (j,k) \in \mathcal{J} \times \mathcal{K}$$

where $X_j^k = M_j^k$ or $X_j^k = O_j^k$.

Let $J = |\mathcal{J}|$ and $K = |\mathcal{K}|$, we now extend the definition of $\widetilde{M}_j^k$ and $\widetilde{O}_j^k$ to $\mathbb{N}^2$ thanks to J periodicity for the degrees and K periodicity for the orders.

Let $\widetilde{L}_j^k$ be defined on $\mathcal{J} \times \mathcal{K}$ by:

$$\widetilde{L}_j^k = \sum_{n \in \mathcal{J}} \sum_{l \in \mathcal{K}} \widetilde{M}_n^l \widetilde{O}_{j-n}^{k-l} \qquad \forall (j,k) \in \mathcal{J} \times \mathcal{K}.$$

We have now the following lemma.

Lemma 2.1.

$$\widetilde{L}_j^k = L_j^k, \qquad \forall (j,k) \in \Gamma$$

Proof. This is due to:

$$\begin{aligned} M_j^k &= 0 \qquad \forall (j,k)/|k| > j, \\ M_j^k &= 0 \qquad \forall (j,k)/j < 0, \\ O_j^k &= 0 \qquad \forall (j,k)/j > P. \end{aligned}$$

□

But $\widetilde{L}_j^k$ can be non-null for $(j,k) \notin \Gamma$.

Once again, we redefine hereafter L_j^k, M_j^k and O_j^k respectively by $\widetilde{L}_j^k$, $\widetilde{M}_j^k$ and $\widetilde{O}_j^k$, and we finally obtain:

$$L_j^k = \sum_{n \in \mathcal{J}} \sum_{l \in \mathcal{K}} M_n^l O_{j-n}^{k-l} \qquad \forall (j,k) \in \mathbb{N}^2.$$

Remark 2.1. *The zero-padding used in the definition of $\mathcal{J}$ results in adding 100 % of zero in the degree dimension. We cannot add less because of the "uncentered" type of the convolution (i.e. n ranges from 0 to $P-j$ in (2.1) whereas a centered convolution have index ranging from $-N+1$ to N). However for the definition of $\mathcal{K}$, i.e. in the order dimension, we do not need zero-padding (because of the nullity out of Γ).*

2.1.2 Use of symmetry properties

The property (1.5), valid for L_j^k and M_j^k as well, will result in Fourier space, with a $1D$ DFT on the orders, in half of the terms being equal to the complex conjugate of the other half.

Indeed when applying a $1D$ DFT on a sequence $(X)^{l \in \mathbb{N}}$, N periodic, with $X^{-l} = (-1)^l \overline{X}^l$, we have:

$$\widehat{X}^{-l} = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} X^k e^{-\frac{i2\pi k(-l)}{N}},$$

$$\widehat{X}^{-l} = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} (-1)^k \overline{X^{-k}} e^{\frac{i2\pi kl}{N}}.$$

Let $X'^n = X^{-n}, \forall n \in \mathbb{N}$.

$$\widehat{X}^{-l} = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \overline{X'^k} e^{\frac{i2\pi k}{N}\left(l+\frac{N}{2}\right)}$$

$$\widehat{X}^{-l} = \overline{\frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} X'^k e^{-\frac{i2\pi k}{N}\left(l+\frac{N}{2}\right)}}$$

That is to say :

$$\widehat{X}^{-l} = \overline{\widehat{X'}^{l+\frac{N}{2}}}.$$

Since: $\widehat{X'}^{k} = \widehat{X}^{-k}$, we have:

$$\widehat{X}^{-l} = \overline{\widehat{X}^{-l-\frac{N}{2}}}.$$

As $(\widehat{X}^k)_{k\in\mathbb{N}}$ is periodic with period N, we can deduce:

$$\widehat{X}^{-l} = \overline{\widehat{X}^{\frac{N}{2}-l}}.$$

That's why only one half of the array containing the terms in Fourier space need to be stored.

2.2 Fast Fourier Transform with blocks

2.2.1 Numerical instability

Due to definition 1.1 and equations (1.3) (1.4), the O_j^k and the M_j^k grow, when j and k grow, as:

$$O_j^k(r,\theta,\phi) \sim \frac{(j-|k|)!}{r^{j+1}} \tag{2.3}$$

$$I_j^k(r,\theta,\phi) \sim \frac{r^j}{(j+|k|)!}. \tag{2.4}$$

As mentioned in [Ell95] this fast variation in the magnitude of the norm of the terms results in numerical instability when using FFT above a critical value for P. Indeed when performing a DFT the terms with small magnitude are added to terms with high magnitude and hence loose their precision: when the backward DFT is performed the information contained in the small magnitude terms is no longer noticeable.

This problem depends on the machine precision used: with floating point computations performed in single precision (4 bytes) the numerical instabilities appear for much lower values of P than with floating point computations performed in double precision (8 bytes).

In double precision this problem does not appear with the classic $M2L$ (i.e. with no FFT) for common values of P (at least up to $P = 40$) since terms are rather multiplied before being added. However in single precision problems can appear with classic $M2L$ but for values of P much higher than with the FFT improvement.

This issue has partly (see sections 2.6 and 2.7) been resolved in [Ell95] with the use of a *block* FFT: in the degrees dimension the terms are grouped according to their degrees, and hence according to the magnitude of their norm, and the DFT is performed for every block seperately. The point-wise product has then to be performed block by block while using a *large grain convolution* (named as in [Ell95]) among the blocks.

We explain below the details of this block FFT.

2.2.2 Block decomposition

We denote:

- B: the number of blocks
- T: the "size" of the block (in the degree dimension, without zero-padding, see below)

We suppose first that $P+1$ is a multiple of T and we have thus: $B = \frac{P+1}{T}$. The block version of the FFT deals only with the degrees dimension, the FFT in the order dimension being unchanged. We therefore consider a $1D$ correlation as:

$$L_j = \sum_{n=0}^{P-j} M_n O_{j+n}.$$

L_i, M_i and O_i being defined on: $[\![0, P]\!]$.

For all $I \in [\![0, B-1]\!]$ we define the block $M(I)$ by:

$$M(I)_i = \begin{cases} M_{IT+i} & \forall i \in [\![0, T-1]\!], \\ 0 & \forall i \in [\![-(T-1), 0]\!]. \end{cases}$$

We extend this definition to $\mathbb{N}$ thanks to $(2T-1)$ periodicity. The same is done for $O(I)$, also extended to $\mathbb{N}$ with $(2T-1)$ periodicity. For all $(I,J) \in [\![0, B-1]\!]^2$, and for all $i \in [\![-(T-1), T-1]\!]$, we introduce the block $X(I,J)$ by:

$$X(I,J)_i = \sum_{n=-(T-1)}^{T-1} M(I)_n O(J)_{i+n}. \tag{2.5}$$

Due to the definition of blocks $M(I)$ and $O(J)$, the sum can be shortened as: $X(I,J)_i = \sum_{n=0}^{T-1-i} M(I)_n O(J)_{i+n}$.

Block $X(I,J)$ is thus the result of the 1D correlation of size $2T-1$ between block $M(I)$ and block $O(J)$. Due to the definition of these blocks (periodicity and zero-padding), we can obtain exactly the same result with the " DFT / inversed point-wise product in Fourier space / backward DFT " scheme.

We now wish to retrieve the L_j coefficients out of the $X(I,J)$ blocks. The FFT for $M2L$ is however not really decomposable into different blocks (as a matrix-product is, for example) since the original result is widespread among different blocks: each L_j is computed as a sum over terms in the $X(I,J)$ blocks. In order to determine which terms to sum for a given L_j we first point out that for all $i \in [\![-(T-1), T-1]\!]$, $X(I, I+J)_i$ is part of the sum for L_{JT+i}.

More precisely:

block $X(I, I+J)$ "interacts" with

$L_{JT-(T-1)}$
$L_{JT-(T-2)}$
$\dots$
L_{JT-1}
L_{JT}
L_{JT+1}
$\dots$
$L_{JT+(T-1)}$

When seen from L_j, since:

$$\forall i \in [\![0, P]\!], \exists! (I, i') \in [\![0, B-1]\!] \times [\![0, T-1]\!] \text{ such that } i = IT + i',$$

we have if $i' = 0$:

$$L_i = \sum_{N=0}^{B-1-I} X(N, N+I)_{i'}, \tag{2.6}$$

and otherwise (i.e. $i' \neq 0$) :

$$L_i = \sum_{N=0}^{B-1-I} X(N, N+I)_{i'} + \sum_{N=0}^{B-1-I-1} X(N, N+I+1)_{i'-T}. \tag{2.7}$$

As in [Ell95] a large-grain convolution (here a correlation) appears at the blocks level in the equations (2.6) and (2.7): this can can be pictured as in figure 2.1.

When $P+1$ is not a multiple of T, the number of blocks used is $\frac{P+1}{T} + 1$ (integer division). The last block has then null rows in addition to those used for zero-padding.

In [ED95] the block version enables the use of shorter blocks in the order dimension for the first blocks: this has not been used here since all FFT computations are precomputed (see section 1.1), which reduces the gain of such enhancement.

2.3 Implementation details

We use FFTW [FJ05] for the efficient implementation of a DFT. This avoids the writing of our own efficient FFT (Fast Fourier Transform) but it has the drawback not to allow the direct use of the symmetries on the orders (see (2.1.2)): we need a full array for the order dimension (i.e. with negative and positive orders) for FFTW while a hand-made special FFT would require an halved array.

More precisely we cannot use the $2D$ FFT of FFTW but we have to split the $2D$ FFT in two sets of $1D$ FFT: the $1D$ FFT in the orders dimension are first performed on a full array which contain both positive and negative orders, and then only one half of this array is kept for the $1D$ FFT in the degrees dimension. After that, the point-wise product is performed with halved arrays and the backward DFT take place in the reverse order.

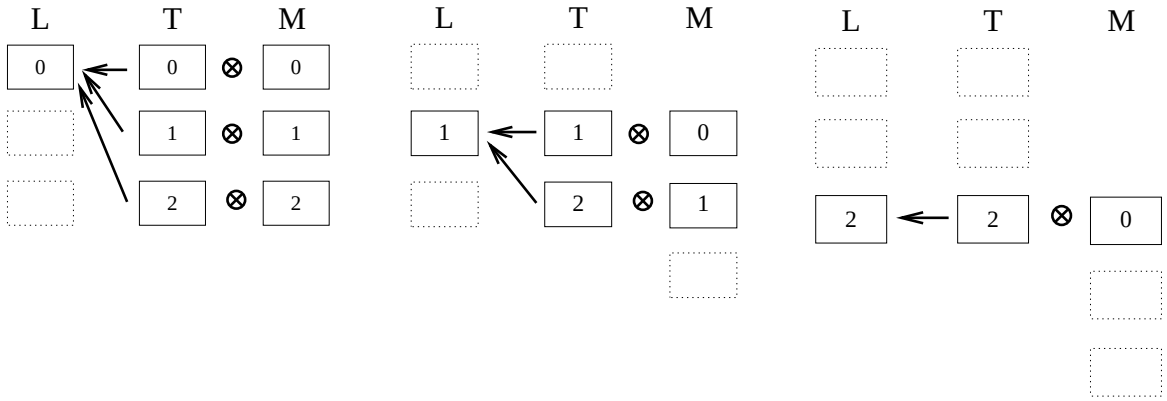


Figure 2.1: Large grain correlation of block FFT. Here M and L denote the blocks containing the multipole and local expansions and T the blocks containing the $M2L$ transfer function.

This implies some recopies from full to halved arrays when performing FFT, but this additional cost is small. The main advantage of FFTW, other than its efficiency, is that it can handle arbitrary sizes for FFT: we are here not restrained to power of 2.

2.3.1 Without blocks

With FFTW a 1D FFT of size N computes the array Y_k:

$$Y_k = \sum_{j=0}^{N-1} X_j e^{\frac{-2i\pi jk}{N}} \qquad \forall k \in [\![0, N-1]\!].$$

Up to now we have redefine our expansion terms on $\mathcal{J} \times \mathcal{K} = [\![-P, P]\!] \times [\![-P, P]\!]$ which are centered around 0: in order to use correct arrays as input of FFTW, the $P+1$ first terms in each of the 2 dimensions have to correspond to the terms with positive (or null) indexes while the P following terms correspond, due to the periodicity used, to the negative indexes. This is the *wrap-around order*.

For a "theoretical" layout of the following expansion, with $P = 4$ (dimensions: $2P+1 \times 2P+1$)

				X_0^0				
			X_1^{-1}	X_1^0	X_1^1			
		X_2^{-2}	X_2^{-1}	X_2^0	X_2^1	X_2^2		
	X_3^{-3}	X_3^{-2}	X_3^{-1}	X_3^0	X_3^1	X_3^2	X_3^3	
X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4

k=−4..4 (orders)

$j = -4..4$ (degrees)

We therefore have rather this storage in memory:

X_0^0								
X_1^0	X_1^1							X_1^{-1}
X_2^0	X_2^1	X_2^2					X_2^{-2}	X_2^{-1}
X_3^0	X_3^1	X_3^2	X_3^3			X_3^{-3}	X_3^{-2}	X_3^{-1}
X_4^0	X_4^1	X_4^2	X_4^3	X_4^4	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}

k=0..8 (orders)

$j = 0..8$ (degrees)

However due to the redefinition of our O_j^k in 2.2, the degree of the $M2L$ transfer function are inversed which gives:

X_0^0								
X_4^0	X_4^1	X_4^2	X_4^3	X_4^4	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}
X_3^0	X_3^1	X_3^2	X_3^3			X_3^{-3}	X_3^{-2}	X_3^{-1}
X_2^0	X_2^1	X_2^2					X_2^{-2}	X_2^{-1}
X_1^0	X_1^1							X_1^{-1}

k=0..8 (orders)

$j = 0..8$ (degrees) (2.8)

Null rows can here clearly be skipped when performing the FFT in the orders dimension (before the FFT in the degrees dimension), but this layout is not favorable when "planing" the FFT with FFTW.

That's why we have choosed not to use wrap-around order: this implies a shift in the indexes for the 2 dimensions (orders and degrees) in real space. With even sizes, the shift equals the half of the dimension and in Fourier space such a shift results in each term being multiply by $(-1)^{j+k}$. And these factors are then canceled during the point-wise product!

But in order to have even sizes we must add a null row and a null column, and the sizes of our FFT are now $2(P+1)$ in each dimension.

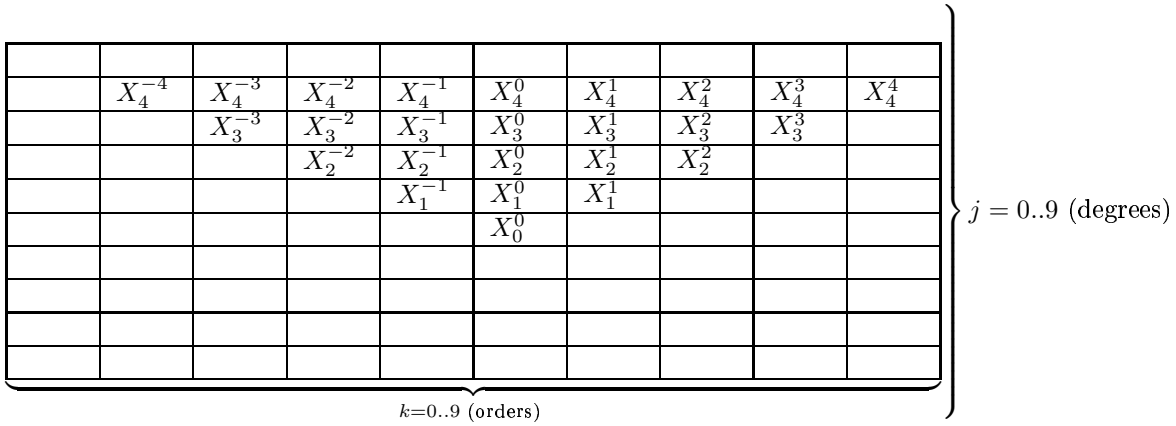
We now have, without wrap-around order and without degree inversion, (i.e. for M_j^k):

					X_0^0				
				X_1^{-1}	X_1^0	X_1^1			
			X_2^{-2}	X_2^{-1}	X_2^0	X_2^1	X_2^2		
		X_3^{-3}	X_3^{-2}	X_3^{-1}	X_3^0	X_3^1	X_3^2	X_3^3	
	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4

k=0..9 (orders)

$j = 0..9$ (degrees)

and with degree inversion (i.e. for O_j^k):



L_j^k obtained with backward FFT are stored with wrap-around order and they have inverted degrees and orders (see equation (2.2)): they are stored in the same way as (2.8) but with inverted orders. And when performing backward FFT in the order dimension we perform the 1D backward FFT only on the rows that will be useful, i.e. the rows that contains degrees j with $0 \leq j \leq P$.

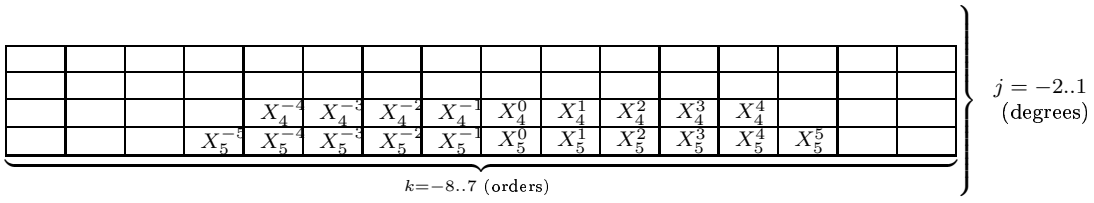
Remark 2.2. *The choice not to use the wrap-around order, for multipole expansion as well as for $M2L$ transfer function, was done in order to reduce the FFT runtime. However since the most time-consuming part is the point-wise product (see section (2.4) , it could have been better to keep the wrap-around order (and hence odd sizes as $2P+1 \times 2P+1$ instead of $2(P+1) \times 2(P+1)$) in order to have smaller array sizes for the point-wise product.*

2.3.2 With blocks

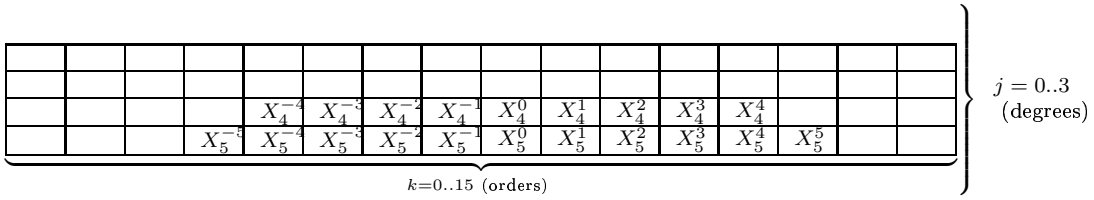
We have presented in section (2.2) a block decomposition of a $1D$ correlation. When applied to $M2L$ operation the order dimension treatment remains unchanged compared to the non-block version. For the degree dimension, we consider a correlation as in (2.1), thus performing a large-grain correlation, but when considering (2.5) we perform the same inversion as (2.2) inside each block: (2.5) are hence converted into a convolution, thus resulting in a non-inversed point-wise product in Fourier space.

We give here examples of the practical layout in memory for our blocks when considering $P = 7$: while a non-block FFT would use arrays of size 16×16, a block FFT, with $T = 2$ and with zero-padding ($T \rightarrow 2T$), has blocks of size 4×16. We represent here block $\#2$.

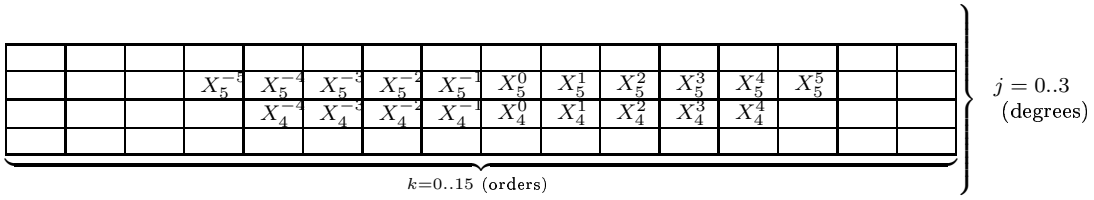
The theoretical layout would be:



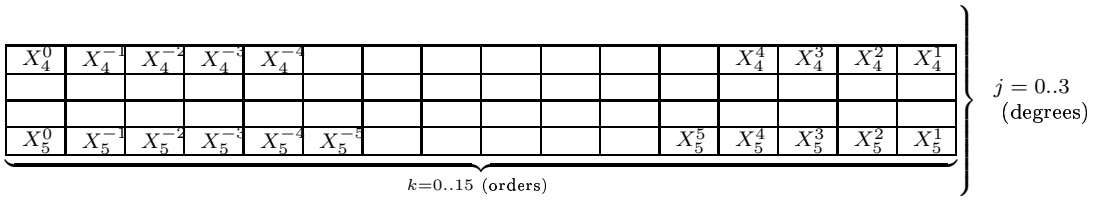
For the multipole expansion (i.e. without wrap-around order, and without inversion):



For the transfer function (i.e. without wrap-around order, and with degrees inversion):



For the local expansion (i.e. with wrap-around order, and with orders and degrees inversion):



As for the non-block version of the FFT, null rows are skipped when performing the forward FFT in the order dimension, for the blocks of the multipole expansion as well as for the blocks of the transfer function. However for the backward FFT in the order dimension (for the blocks of the local expansion), there is only 1 "useless" rows per block: all the other rows contain useful terms since the block decomposition spreads the local expansion terms inside each block. In practice we do not skip any row in this case.

Remark 2.3 (BLAS usage study). *The point-wise product, in block version as in non-block version, which is the most time-consuming part of the $M2L$ computation with the FFT improvement, does not match any of the standard BLAS calls (see section 4). In fact not much speedup can be obtained from such computation since there is no data reuse: once the single loop has been written it is up to the compiler to provide code that can be pipelined.*

2.4 Complexity

We now study the complexity of the block and the non-block version of the FFT enhancement for $M2L$. First we consider that the $M2L$ transfer functions (O_j^k) are precomputed for each level l and their conversion to Fourier space too. For growing l, the runtime cost of this precomputing becomes quickly insignificant. Secondly, the $M2L$ computation with FFT proceeds like this: all multipole expansions are first converted to Fourier space, then for each cell we add the results of all point-wise products in its interaction list, and convert this addition to real space in order to retrieve its local expansion. As the size of the interaction list is 189 (for $ws = 1$), we consider:

- 1 conversion to Fourier space for a multipole expansion
- 189 point-wise products
- 1 backward conversion to real space for a local expansion

This, when multiply by the number of cells at level l, gives a precise estimate of the number of operations needed for all $M2L$ at level l. The estimated operation count used for a $1D$ complex FFT of size N with FFTW is: $5N\log_2(N)$ (see online FFTW documentation at: *http://www.fftw.org*).

2.4.1 Without blocks

Operation count 2.1. *For 189 $M2L$ operations, the theoretical operation count of the FFT without block is:*

$$189\ M2L = 40(P+1)^2\log_2(2P+2) + 378 * (P+1)^2.$$

Proof. With the use of symmetries for orders (as exposed in (2.1.2)), the use of an extra row and column in order to have even sizes like $2(P+1) \times 2(P+1)$ (see section (2.3.1)), and $1D$ (forward and backward) FFT being performed only on the $P+1$ "useful" rows, we have for the non-block version:

- for FFT on multipole expansion array, we count $P+1$ FFT of size $2(P+1)$ in the order dimension (full array with null rows skipped) and $P+1$ FFT of size $2(P+1)$ in the degree dimension (halved array), i.e.:

$$\begin{aligned}&(P+1)5(2P+2)\log_2(2P+2) + (P+1)5(2P+2)\log_2(2P+2)\\ &= 20(P+1)^2\log_2(2P+2),\end{aligned}$$

- for backward FFT on local expansion array, we count $P+1$ FFT of size $2(P+1)$ in the degree dimension (halved array) and then $P+1$ FFT of size $2(P+1)$ in the order dimension (full array with useless rows skipped), i.e.:

$$20(P+1)^2\log_2(2P+2),$$

- 189 point-wise products between halved arrays (of size $2(P+1) \times (P+1)$):

$$189 * 2 * (P+1)^2 .$$

□

For usual values of P, the most time-consuming part of the FFT version of $M2L$ is thus the point-wise product.

2.4.2 With blocks

Operation count 2.2. *For* 189 $M2L$ *operations, the theoretical operation count of the FFT with block is, with* $B = \frac{P+1}{T}$*:*

$$189\ M2L = 10\,(P+1)^2 \log_2\left(32\,(P+1)^2\,T^2\right) + 189\left(\frac{(P+1)^3}{T} + (P+1)^2\right).$$

Proof. We count the same operations as for the non-block version and we here suppose that $P+1$ is a multiple of T. The size of a block with zero-padding is then $2T \times 2(P+1)$ for FFT in the order dimension (full array), and $2T \times (P+1)$ for the FFT in the degree dimension and the large-grain correlation (halved arrays).

- For FFT on the B multipole expansion blocks, we count T FFT of size $2(P+1)$ in the order dimension (full array with null rows skipped) and $P+1$ FFT of size $2T$ in the degree dimension (halved array), i.e.:

$$\begin{aligned} &B\,(5T(2P+2)\log_2(2P+2) + (P+1)5(2T)\log_2(2T)) \\ &\quad = 10BT(P+1)\log_2(4(P+1)T). \end{aligned}$$

- For backward FFT on the B local expansion blocks, we count $P+1$ FFT of size $2T$ in the degree dimension (halved array) and then $2T$ FFT of size $2(P+1)$ in the order dimension (full array but no blank row skipped), i.e.:

$$\begin{aligned} &B\,(5(P+1)(2T)\log_2(2T) + (2T)5(2P+2)\log_2(2P+2)) \\ &\quad = 10BT(P+1)\log_2(8(P+1)T). \end{aligned}$$

- 189 large grain correlations with point-wise products between blocks of size $2T \times (P+1)$:

$$189\left[\sum_{J=0}^{B-1}\sum_{I=0}^{B-1-J}(2T\,(P+1))\right] = 189B(B+1)T(P+1).$$

We thus finally have:

$$189\ M2L = 10\,(P+1)^2 \log_2\left(32\,(P+1)^2\,T^2\right) + 189\,(P+1)^2\,(B+1)\,.$$

□

We here see that the block version of the FFT for $M2L$ has an operation count in $\mathcal{O}(P^3)$, against $\mathcal{O}(P^2 \log_2 P)$ for the non-block version. We can also remark that the large-grain correlation / point-wise product part is even more time-consuming than the FFT / backward FFT part, compared to the non-block FFT.

2.5 Memory requirements

We now compare the memory needed by the FFT improvement with the needs of the classic $M2L$ computation as exposed in section 1.2.3. With the same notations we measure the needs for the non-block version. The block version has the same memory requirements when $P+1$ is a multiple of T. Otherwise the size used is the same as for the first greater multiple of T (minus 1).

Since the expansions in Fourier space are stored in halved arrays with complex numbers, their size is $2(P+1)^2 c$. This is valid for multipole and local expansions as well as for the transfer function. However we only need one single halved arrays for the local expansions: this is used as an auxiliary array where all $M2L$ results of the interaction list are accumulated for the current cell being processed. The size of this single array is neglected.

We have thus (here for single height kernel):

$$Mem_{Sg}(P,H) = \mathcal{N}(H)\left(2(P+1)^2 + \frac{(P+1)(P+2)}{2}\right)c + \mathcal{N}_{\mathcal{T}}\left(2(P+1)^2\right)c. \tag{2.9}$$

2.6 Remaining instability

However tests have shown that the numerical instability remains for high values of P as shown in figure 2.2 for single height $M2L$ kernel. This might be explain by the influence of the value of the order (i.e. k) in (2.3) and in (2.4): the block decomposition is only performed for the degree dimension, but for high values of P, numerical instability appears in the order dimension too.

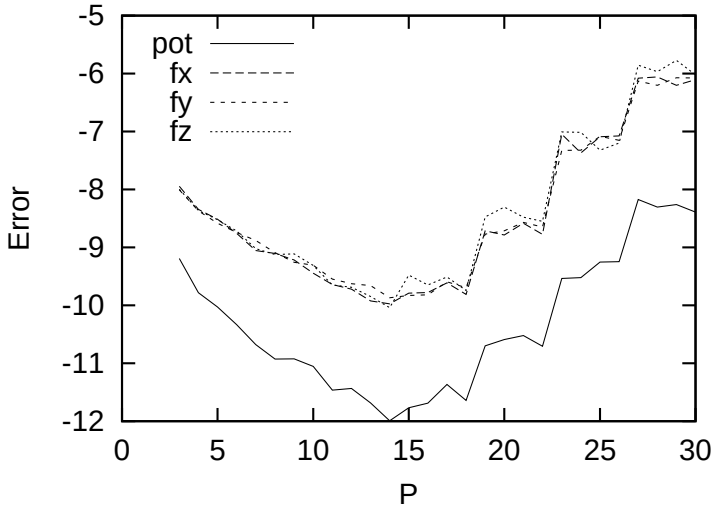


Figure 2.2: Logarithmic error for the potential (pot) and the force coordinates (fx, fy, fz) according to P for single height $M2L$ kernel with FFT with blocks of size 4. Tests performed on 10000 uniform distribution with an octree of height 4. The error plotted is the maximum absolute error over all the particles.

This numerical instability can however be reduced with lower block size: while, for single height $M2L$ kernel, the FFT with block size 4 becomes unstable for P higher than 14 (see figure 2.2), a FFT with block size 3 is stable until 27, but is of course slower.

Another severe drawback is that the numerical instability increases with the height of the octree. When considering (2.3) and (2.4), we have also to take into account the influence of two distances:

- the distance between 2 cell centers (for O_j^k in $M2L$ for example),
- and the distance between a cell center and a particle location (for M_j^k in $P2M$).

With increasing height of the octree, these distances decrease, since the cell size is divided by 2 between two consecutive levels of the octree, hence resulting in numerical instability when their order of magnitude is becomming too small. For example, as seen in figure 2.3, with an octree of height 6, the FFT with block size 4 is unstable for P higher than 10 (compared to 14 for height 4).

The values of P and heights above which numerical instabilities appear are summarized in table[3] 2.10 for single height $M2L$ kernel (for FFT of size 2, no influence of the octree height on the instabilities was detected, but this will probably happen for higher heights that cannot be run here in sequential computation). Of course these values depend on the size of the computational box that encloses all particles: our tests are performed with all particle coordinates inside $[0, 1.0]$ but with a bigger computational box the numerical instabilities would appear for higher heights. Moreover the instability due to increasing octree height can be avoided with a simple trick: when multiplying all particle cartesian coordinates, as well as the coordinates of the computational box, with a scalar $\alpha > 1$, the two kinds of distances mentioned above are then increased which can reduce the numerical instability. The original potential and forces can be obtained afterward thanks to a multiplication with respectively $\frac{1}{\alpha}$ and $\frac{1}{\alpha^2}$. For example with $\alpha = 4$, the numerical instabilities with an height of 6 correspond to the ones formerly obtained with an height of 4. And with $\alpha = 4$ and an height of 4, no instabilities are detected in our tests for P between 3 and 40 with block size 4. This drawback however still exists and restrains the use of the FFT improvement for the FMM: the critical leaf size is a priori not known and depends on the

[3] Tests run on IBM Power3 (2 GB) and Power4 (8GB) at the LaBRI, as well as on IBM Power3 (16 GB) and Power4 (32 GB) at the CINES. See section 4 for more details on these machines.

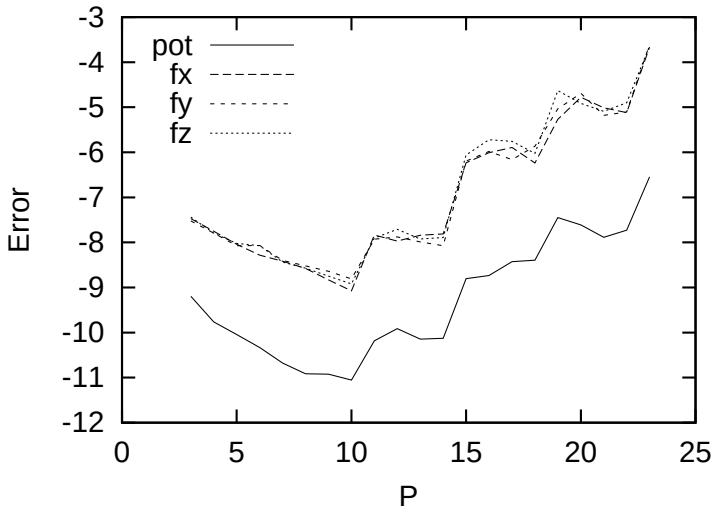


Figure 2.3: Logarithmic error for the potential (pot) and the force coordinates (fx, fy, fz) according to P for single height $M2L$ kernel with FFT with blocks of size 4. Tests performed on 10000 uniform distribution with an octree of height 6. The error plotted is the maximum absolute error over all the particles.

machine precision used. Moreover whether this scaling has to be used or not depends on the value of P, the block size and the leaf size.

	$height = 4$	$height = 5$	$height = 6$
FFT with block size 4	14	14	10
FFT with block size 3	27	23	20
FFT with block size 2	40	40	40

(2.10)

2.7 Comparison with DPMTA code

In order to check the exactness and the efficiency of our FFT implementation for the FMM, we have compare our code (FMB) with DPMTA (version 3.1.2p3). We have been using DPMTA with the fixed interaction list of Greengard & Rokhlin with $ws = 1$: i.e. no MAC (Multipole Acceptance Criteria, or θ) was used. As in our code, all $M2L$ transfer functions were precomputed. And the P_{DPMTA} used in DPMTA differs from our: $P_{DPMTA} = P + 1$. Comparison was performed on IBM Power3 and Power4, thus requiring a few modifications in the Makefiles so that it compiles with the IBM C compiler (xlc). The same compiler options were used for DPMTA and FMB (namely for Power3: *xlc -O3 -qstrict -bmaxdata:0x80000000 -qarch=pwr3 -qtune=pwr3 -Q=150 -qstaticinline*). Figure 2.4 shows that our implementation of the FFT improvement for the FMM is as fast as the one DPMTA.

Moreover we have also found numerical instabilities when testing DPMTA on full FMM computations: these appear for the same values of P and heights as with FMB (see section 2.6).

2.8 Fast Fourier Transform for double height $M2L$ kernel

With double height $M2L$ kernel, the maximum degree of the O_j^k is $2P$. We therefore need $\mathcal{J}$ and $\mathcal{K}$ to be defined by: $\mathcal{J} = [\![-2P, 2P]\!]$ and $\mathcal{K} = [\![-2P, 2P]\!]$. But thanks to the use of symmetries (see (2.1.2)) we actually use: $\mathcal{K} = [\![0, 2P]\!]$.

The same $\mathcal{J}$ and $\mathcal{K}$ have to be used for M_j^k and L_j^k: this is obtained with zero-padding. This done, the FFT for double height $M2L$ kernel operates in the same way than for single height.

With double height $M2L$ kernel, the numerical instability is even worst than with single height, and the need for block decomposition of the FFT is higher: since O_j^k have degrees up to $2P$, the range of magnitude is even greater and they become unstable for even lower values of P. As for FFT without block, it is rather straightforward to derive block FFT for double height kernel from block FFT for single height kernel once $\mathcal{J}$ and $\mathcal{K}$ have been defined. For a same T, we have roughly twice more blocks with double height kernel than with single height kernel.

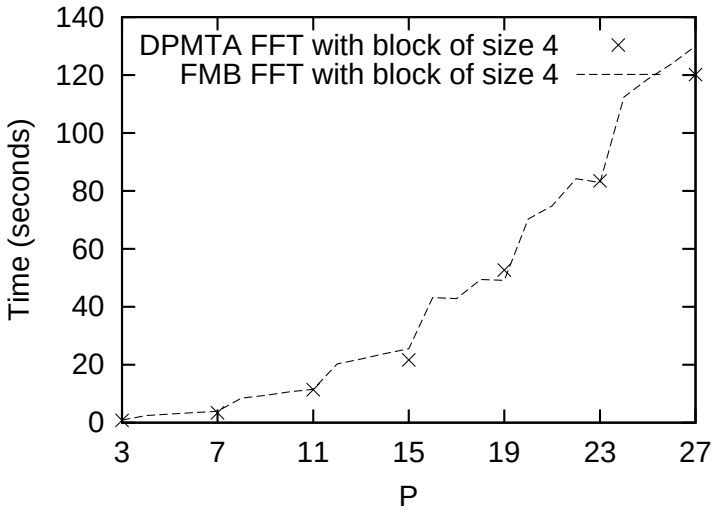


Figure 2.4: Downward pass CPU times for DPMTA and FMB: the octree height is 4 and the uniform distribution contains 100000 so that there is no empty leaf (these are skipped by DPMTA). All particle coordinates were within $[0, 1.0]$. We use single height $M2L$ kernel as imposed by DPMTA, and blocks of size 4 for FFT. The P plotted is the one of FMB ($P_{DPMTA} = P + 1$) and DPMTA accept for P_{DPMTA} only the multiples of the block size.

But even with the block version, the range of values for P that lead to correct computations for double height kernel is smaller than in the single height kernel case (unless some rescaling is performed as explained in section 2.6). For comparison with table 2.10, block FFT with block size 4 is here only stable for $P \leq 8$ (in an octree of height 4 and a computational box of side 1.0). When trading runtime in for accuracy, we can reach 12 with block size 3, and 19 with block size 2. And as for single height kernel, this is even worst for growing heights of the octree.

In order to establish the complexity for the double height kernel, we consider the same operations as in section (2.4).

For the non-block version, we just have to replace P by $2P$:

$$189\ M2L = 40\,(2P+1)^2 \log_2\left(2\,(2P+1)\right) + 378 * (2P+1)^2\,.$$

Remark 2.4. *In practice, more than $2P+1$ rows can be skipped for FFT on multipole expansion arrays and backward FFT on local expansion arrays since the maximum degree of these expansions is P.*

For the block version of the FFT, we consider that $2P+1$ is a multiple of T: this is certainly never the case for $T = 4$ for example, but the behaviour so described is close to reality. We replace then P by $2P$ and the number of block is $B = \frac{2P+1}{T}$. The size of a block is now either $2T \times 2(2P+1)$ (full array) or $2T \times (2P+1)$ (halved arrays), and we have:

$$189\ M2L = 10\,(2P+1)^2 \log_2\left(32\,(2P+1)^2\,T^2\right) + 189\left(\frac{(2P+1)^3}{T} + (2P+1)^2\right).$$

Remark 2.5. *As for the non-block version, more rows can be skipped in practice.*

Finally we give the memory requirements for the double height kernel, proceeding in the same way as in section 2.5. The size of all halved arrays is now $2(2P+1)^2$, for multipole expansions and transfer functions, and thus:

$$Mem_{Db}(P,H) = \mathcal{N}(H)\left(2(2P+1)^2 + \frac{(P+1)(P+2)}{2}\right)c + \mathcal{N}_{\mathcal{T}}\left(2(2P+1)^2\right)c. \tag{2.11}$$

2.9 Comparison between single and double height kernels

We first recall that the ratio between single and double height kernels with classic $M2L$ computation is roughly 6: see section 1.2.4.

For the FFT without blocks, we have, without considering the log_2, a ratio of 4 which favors the double height kernel. However due to its higher numerical instability, the double height kernel imposes even more the block version. And for a block version of the FFT, the ratio is still 4 for the FFT / backward FFT parts but it equals roughly 8 for the large-grain correlation / point-wise product part: this clearly does not favor the double height kernel.

3 Rotations

The use of *rotations* has already been introduced in several articles: see [CGR99], [CIGR99], [GR97], [GD03] and [WHG96]. This improvement allows the computation of $M2M$, $M2L$ and $L2L$ operators in $\mathcal{O}(P^3)$, against $\mathcal{O}(P^4)$ for their classic version.

We have chosen to use the formulae detailed by Gumerov and Duraiswami in [GD03] since: they use the same definition for spherical harmonics, they use symmetries to speed up the computation and they focus only on the needed rotations. Moreover the recurrence is performed on real numbers (and not complex ones) and is simple to initiate. However no proof is given on the numerical stability of the formulae used.

We here present the principle of this scheme and the formulae used for both single and double height $M2L$ kernels. We give also tailored formulae for the Outer (O_j^k) and Inner (I_j^k) functions. Finally, we will outline the fact that the $M2L$ computation with rotations does not match standard BLAS calls and we will present numerical stability checking.

3.1 Formulae

The general scheme for $M2L$ computation with rotations is pictured in figure 3.1. We will here introduce the required mathematical formulae and detail this scheme.

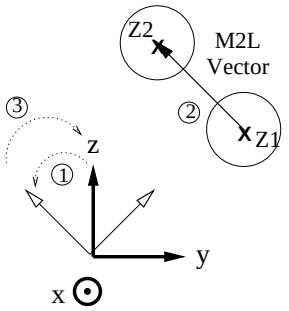


Figure 3.1: $M2L$ with rotations. 1: rotation of the coordinate system so that the $M2L$ vector is along the **z** axis. 2: $M2L$ performed along the **z** axis. 3: inverse rotation back to the original coordinate system.

3.1.1 Methodology

Notations. We denote $\mathcal{R}_x(\alpha)$, $\mathcal{R}_y(\alpha)$, $\mathcal{R}_z(\alpha)$ the rotations of the Cartesian system of coordinates by angle α about respectively **x**, **y** **z** axes: rotations are considered to be performed in the counterclockwise direction, when looking towards the origin from a location with positive coordinate on the corresponding axis of rotation.

Given a point **P** with spherical coordinates (r, θ, ϕ) in the original coordinate system, the spherical coordinates in the rotated system are denoted: $(r, \widehat{\theta}, \widehat{\phi})$.

Generally, we first consider a rotation (of the Cartesian coordinate system) by angle χ about **z** axis, followed by a rotation by angle γ about the **y** axis, and finally a rotation by angle $\pi - \omega$ about the new **z** axis, i.e.: $\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$.

Remark 3.1. *In [GD03] the second rotation is performed about the* **x** *axis.*

Rotation of spherical harmonics. As shown in the cited papers, for the rotations $\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$ there exists coefficients $T_j^{\nu,k}(\omega, \gamma, \chi)$, with $j \geq 0, |k| \leq j, |\nu| \leq j$, such that the spherical harmonics $Y_j^k(\theta, \phi)$ in the original coordinate system can be written with spherical harmonics in the rotated system as:

$$Y_j^k(\theta, \phi) = \sum_{\nu=-j}^{j} T_j^{\nu,k}(\omega, \gamma, \chi) Y_j^{\nu}(\widehat{\theta}, \widehat{\phi}). \tag{3.1}$$

The same $T_j^{\nu,k}$ apply on both normalized and unnormalized spherical harmonics.

The Outer and Inner functions, as defined in (1.3) and in (1.4), being based on these spherical harmonics, there exists also coefficients $T_{O_j}^{\nu,k}(\omega,\gamma,\chi)$, and $T_{I_j}^{\nu,k}(\omega,\gamma,\chi)$ such that:

$$O_j^k(r,\theta,\phi) = \sum_{\nu=-j}^{j} T_{O_j}^{\nu,k}(\omega,\gamma,\chi) O_j^\nu(r,\widehat{\theta},\widehat{\phi}), \tag{3.2}$$

and:

$$I_j^k(r,\theta,\phi) = \sum_{\nu=-j}^{j} T_{I_j}^{\nu,k}(\omega,\gamma,\chi) I_j^\nu(r,\widehat{\theta},\widehat{\phi}). \tag{3.3}$$

These coefficients $T_j^{\nu,k}$, $T_{O_j}^{\nu,k}$ and $T_{I_j}^{\nu,k}$ are considered to vanish for $|\nu| > j$ or $|k| > j$. Hereafter they are named *rotation coefficients*.

Rotation of multipole expansions. Let point $\mathbf{P}$ of spherical coordinates (r,θ,ϕ) in the original coordinate system. The potential evaluated at $\mathbf{P}$ with a multipole expansion is:

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} M_n^l O_n^{-l}(r,\theta,\phi).$$

In the rotated system, the potential writes:

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{\nu=-n}^{n} \widehat{M}_n^\nu O_n^{-\nu}(r,\widehat{\theta},\widehat{\phi}),$$

with the new multipole expansion terms being defined by:

$$\widehat{M}_n^\nu = \sum_{l=-n}^{n} M_n^l T_{O_n}^{-\nu,-l}(\omega,\gamma,\chi). \tag{3.4}$$

Rotation of local expansions. With a local expansion, the potential at $\mathbf{P}$ in the original coordinate system writes:

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} L_n^l I_n^l(r,\theta,\phi).$$

In the rotated system, the potential is given by:

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{\nu=-n}^{n} \widehat{L}_n^\nu I_n^\nu(r,\widehat{\theta},\widehat{\phi}),$$

where the new local expansion terms are defined by:

$$\widehat{L}_n^\nu = \sum_{l=-n}^{n} L_n^l T_{I_n}^{\nu,l}(\omega,\gamma,\chi). \tag{3.5}$$

M2L along the z axis. The improvement in the use of rotations for $M2L$ operator is based on the fact that the cost of an $M2L$ operation performed along the $\mathbf{z}$ axis is $\mathcal{O}(P^3)$ against $\mathcal{O}(P^4)$ for general $M2L$. Indeed the associated Legendre functions verify:

$$P_n^m(1) = \delta_{m,0}$$

where $\delta_{i,j}$ is the Kronecker symbol. We have therefore:

$$O_j^k(r,0,0) = \begin{cases} \frac{(-1)^j}{A_j^0 r^{j+1}} = \frac{j!}{r^{j+1}} & \text{if } k=0, \\ 0 & \text{otherwise,} \end{cases}$$

and:

$$I_j^k(r,0,0) = \begin{cases} A_j^0 r^j = \frac{(-1)^j r^j}{j!} & \text{if } k=0, \\ 0 & \text{otherwise.} \end{cases}$$

When the $M2L$ vector (defined by $\mathbf{z_2} - \mathbf{z_1}$ in definition 1.3) has spherical coordinates in the form $(r, 0, 0)$, the $M2L$ operator can then be computed by:

$$L_j^k = \sum_{n=0}^{+\infty} M_n^{-k} O_{j+n}^0(r,0,0) = \sum_{n=0}^{+\infty} M_n^{-k} \frac{(-1)^{j+n}}{A_{j+n}^0 r^{j+n+1}}. \tag{3.6}$$

Since M_n^{-k} imposes $n \geq |k|$, we have for single height $M2L$ kernel:

$$L_j^k = \sum_{n=|k|}^{P-j} M_n^{-k} \frac{(-1)^{j+n}}{A_{j+n}^0 r^{j+n+1}}, \tag{3.7}$$

and for double height $M2L$ kernel:

$$L_j^k = \sum_{n=|k|}^{P} M_n^{-k} \frac{(-1)^{j+n}}{A_{j+n}^0 r^{j+n+1}}. \tag{3.8}$$

Regular $M2L$ with rotations. We can now detail the scheme pictured in figure 3.1 for a general $M2L$ operation with cell center for multipole expansion $\mathbf{z_1}$ and cell center for local expansion $\mathbf{z_2}$. The $M2L$ vector $\mathbf{V} = \mathbf{z_2} - \mathbf{z_1}$ is considered to have spherical coordinates: (r, θ, ϕ). We first rotate the coordinate system so that the vector $\mathbf{V}$ is along the $\mathbf{z}$ axis. This can be done with the 2 following rotations: $\mathcal{R}_y(\theta).\mathcal{R}_z(\phi)$. However we use a more general scheme with 3 rotations that can be use for the inverse rotation as exposed afterwards. Hence we use $\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$ with: $\omega = \pi$, $\gamma = \theta$, $\chi = \phi$, i.e.:

$$\mathcal{R}_z(0).\mathcal{R}_y(\theta).\mathcal{R}_z(\phi).$$

The coordinates of $\mathbf{V}$ in the new coordinate system are: $(r, 0, 0)$. The new multipole expansion is then given by $\widehat{O}_j^k$ (see (3.4)). With (3.6) we compute $\widehat{L}_j^k$ which represent the local expansion in the rotated coordinate system. In order to obtain the local expansion in the original system, we have to use (3.5) with the inverse rotation:

$$\mathcal{R}_z(\pi - \phi).\mathcal{R}_y(\theta).\mathcal{R}_z(\pi).$$

Indeed the inverse rotation for the general scheme is:

$$\mathcal{R}_z(-\chi).\mathcal{R}_y(-\gamma).\mathcal{R}_z(-(\pi - \omega)) = \mathcal{R}_z(\pi - \chi).\mathcal{R}_y(\gamma).\mathcal{R}_z(\omega). \tag{3.9}$$

In conclusion we need to compute the following rotation coefficients: ${T_O}_j^{\nu,k}(\pi, \theta, \phi)$ and ${T_I}_j^{\nu,k}(\phi, \theta, \pi)$, for all $j \geq 0, |\nu| \leq j, |k| \leq j$.

3.1.2 Computing the rotation coefficients

Most of the following formulae are simply a rewriting of those presented in [GD03]. The complete method used to compute the rotation coefficients is given at the end of this section: all the mathematical background is first presented.

Rotation matrix. Let $\mathbf{V}$ denote a vector with Cartesian coordinates (x, y, z) in coordinate system $(\mathbf{i}_x, \mathbf{i}_y, \mathbf{i}_z)$. After some rotations of the coordinate system, we denote $\widehat{\mathbf{V}} = (\widehat{x}, \widehat{y}, \widehat{z})$ its new coordinates in the rotated coordinate system $(\mathbf{i}_{\widehat{x}}, \mathbf{i}_{\widehat{y}}, \mathbf{i}_{\widehat{z}})$. We then denote $\mathbf{Q}$ the following rotation matrix: $\widehat{\mathbf{V}} = \mathbf{QV}$, i.e.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{i}_{\widehat{x}}.\mathbf{i}_x & \mathbf{i}_{\widehat{x}}.\mathbf{i}_y & \mathbf{i}_{\widehat{x}}.\mathbf{i}_z \\ \mathbf{i}_{\widehat{y}}.\mathbf{i}_x & \mathbf{i}_{\widehat{y}}.\mathbf{i}_y & \mathbf{i}_{\widehat{y}}.\mathbf{i}_z \\ \mathbf{i}_{\widehat{z}}.\mathbf{i}_x & \mathbf{i}_{\widehat{z}}.\mathbf{i}_y & \mathbf{i}_{\widehat{z}}.\mathbf{i}_z \end{bmatrix} \tag{3.10}$$

As exposed in section 3.1.1, we focus on the rotations: $\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$. In this case, we have:

$$\mathbf{Q}(\omega, \gamma, \chi) = \begin{bmatrix} -\cos\omega & \sin\omega & 0 \\ -\sin\omega & -\cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Or:

$$\mathbf{Q}(\omega,\gamma,\chi) = \begin{bmatrix} -\cos\omega.\cos\gamma.\cos\chi - \sin\omega.\sin\chi & -\cos\omega.\cos\gamma.\sin\chi + \sin\omega.\cos\chi & \cos\omega.\sin\gamma \\ -\sin\omega.\cos\gamma.\cos\chi + \cos\omega.\sin\chi & -\sin\omega.\cos\gamma.\sin\chi - \cos\omega.\cos\chi & \sin\omega.\sin\gamma \\ \sin\gamma.\cos\chi & \sin\gamma.\sin\chi & \cos\gamma \end{bmatrix} \tag{3.11}$$

$\mathbf{Q}(\omega,\gamma,\chi)$ is an orthogonal matrix and moreover we have (see equation (3.9)):

$$\mathbf{Q}^{-1}(\omega,\gamma,\chi) = \mathbf{Q}(\chi,\gamma,\omega).$$

Recursive computation of $T_j^{\nu,k}$. We use the same definition of the spherical harmonics Y_j^k (see (A.2)) as in [GD03]. In their paper, Gumerov & Duraiswami use the equation (5.39) (with increasing degrees j for $T_j^{\nu,k}$) as an example for a recursive computation of the $T_j^{\nu,k}$: we here prefer their equation (5.28) with decreasing degrees.

With the matrix $\mathbf{W}$ defined by (with $i=\sqrt{-1}$):

$$\mathbf{W} = \begin{bmatrix} \mathbf{i}_x.(\mathbf{i}_{\widehat{x}} - i\mathbf{i}_{\widehat{y}}) & \mathbf{i}_x.(\mathbf{i}_{\widehat{x}} + i\mathbf{i}_{\widehat{y}}) & -2\mathbf{i}_x.\mathbf{i}_{\widehat{z}} \\ i\mathbf{i}_y.(\mathbf{i}_{\widehat{x}} - i\mathbf{i}_{\widehat{y}}) & i\mathbf{i}_y.(\mathbf{i}_{\widehat{x}} + i\mathbf{i}_{\widehat{y}}) & -2i\mathbf{i}_y.\mathbf{i}_{\widehat{z}} \\ -\frac{1}{2}\mathbf{i}_z.(\mathbf{i}_{\widehat{x}} - i\mathbf{i}_{\widehat{y}}) & -\frac{1}{2}\mathbf{i}_z.(\mathbf{i}_{\widehat{x}} + i\mathbf{i}_{\widehat{y}}) & \mathbf{i}_z.\mathbf{i}_{\widehat{z}} \end{bmatrix}$$

(5.28) in [GD03] is:

$$2b_{n+1}^{-m-1}T_{n+1}^{\nu,m+1} = (W_{1,1}+W_{2,1})b_{n+1}^{-\nu}T_n^{\nu-1,m} + (W_{1,2}+W_{2,2})b_{n+1}^{\nu}T_n^{\nu+1,m} + (W_{1,3}-W_{2,3})a_n^{\nu}T_n^{\nu,m}.$$

With the convention $T_n^{\nu,m} = 0, \forall (n,m)/|m| > n$, we have the following equations (which correspond to (5.30) and (5.33) in [GD03]):

$$2b_{n+1}^{-m-1}T_{n+1}^{n+1,m+1} = (W_{1,1}+W_{2,1})b_{n+1}^{-n-1}T_n^{n,m},$$

$$2b_{n+1}^{-m-1}T_{n+1}^{-n-1,m+1} = (W_{1,2}+W_{2,2})b_{n+1}^{-n-1}T_n^{-n,m}.$$

With (3.10) and (3.11), we obtain the following relationship between $\mathbf{W}$ with $\mathbf{Q}$:

$$W_{1,1}+W_{2,1} = \mathbf{Q}_{1,1}+\mathbf{Q}_{2,2}+i(\mathbf{Q}_{1,2}-\mathbf{Q}_{2,1}) = (\cos\gamma+1)e^{i\chi}e^{i(\pi-\omega)},$$

$$W_{1,2}+W_{2,2} = \mathbf{Q}_{1,1}-\mathbf{Q}_{2,2}+i(\mathbf{Q}_{1,2}+\mathbf{Q}_{2,1}) = (\cos\gamma-1)e^{i\chi}e^{-i(\pi-\omega)},$$

$$W_{1,3}+W_{2,3} = -2\sin\gamma.e^{i\chi}.$$

Hence:

$$T_{n+1}^{\nu,m+1} = \frac{e^{i\chi}}{b_{n+1}^{-m-1}} \left\{\frac{1}{2}\left[(\cos\gamma+1)\,e^{i(\pi-\omega)}b_{n+1}^{-\nu}T_n^{\nu-1,m} + (\cos\gamma-1)e^{-i(\pi-\omega)}b_{n+1}^{\nu}T_n^{\nu+1,m}\right] - \sin\gamma a_n^{\nu}T_n^{\nu,m}\right\}.$$

This recursion is performed in $\mathbb{C}$: we will now simplify it so that it will be held in $\mathbb{R}$.

Recursive computation in $\mathbb{R}$. When performing rotation $\mathcal{R}_z(\alpha)$ on the spherical harmonic $Y_j^k(\theta,\phi)$ we simply have: $Y_j^k(\theta,\phi) = e^{ik\alpha}Y_j^k(\widehat{\theta},\widehat{\phi})$. We can thus decompose $T_n^{\nu,m}(\omega,\gamma,\chi)$ as (once again we refer to [GD03] for details and explicit proof):

$$T_n^{\nu,m}(\omega,\gamma,\chi) = e^{im\chi}e^{-i\nu\omega}H_n^{\nu,m}(\gamma) \tag{3.12}$$

with $H_n^{\nu,m}(\gamma) \in \mathbb{R}$.

The recursive equation for $H_n^{\nu,m}(\gamma)$ is:

$$H_{n+1}^{\nu,m+1} = \frac{1}{\frac{1}{b_{n+1}^{-m-1}}}\left\{\frac{1}{2}\left[-(\cos\gamma+1)\,b_{n+1}^{-\nu}H_n^{\nu-1,m} - (\cos\gamma-1)b_{n+1}^{\nu}H_n^{\nu+1,m}\right] - \sin\gamma a_n^{\nu}H_n^{\nu,m}\right\}. \tag{3.13}$$

The recursion is initiated with $H_n^{\nu,0}(\gamma)$ thanks to the following lemma:

Lemma 3.1.

$$H_n^{\nu,0}(\gamma) = (-1)^\nu \sqrt{\frac{(n-|\nu|)!}{(n+|\nu|)!}} P_n^{|\nu|}(\cos\gamma).$$

Proof. We use the same scheme as [GD03] for their computation. Namely we consider $\mathcal{R}_z(\pi-\omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(0)$ with $\mathbf{V} = (r,\theta,\phi)$. θ is here the angle between $\mathbf{V}$ et $\mathbf{i}_z$: it is also the angle between $\widehat{\mathbf{V}}$ et $\mathbf{i}_{\widehat{z}}$. Moreover the spherical coordinates of $\mathbf{z}$ in the rotated coordinate system $(\mathbf{i}_{\widehat{x}}, \mathbf{i}_{\widehat{y}}, \mathbf{i}_{\widehat{z}})$ are: $(1,\gamma,\omega)$. Applying the Addition Theorem for (unnormalized) spherical harmonics (theorem 5) between $\mathbf{V}$ and $\mathbf{z}$ in the rotated system thus yields:

$$P_n(\cos\theta) = \sum_{\nu=-n}^{n} Y_n^{-\nu}(\gamma,\omega) Y_n^\nu(\widehat{\theta},\widehat{\phi}).$$

We identify this equation with this one (obtained from (3.1)):

$$P_n(\cos\gamma) = Y_n^0(\theta,\phi) = \sum_{\nu=-n}^{n} T_n^{\nu,0}(\omega,\gamma,0) Y_n^\nu(\widehat{\theta},\widehat{\phi})$$

which gives for all χ:

$$T_n^{\nu,0}(\omega,\gamma,\chi) = Y_n^{-\nu}(\gamma,\omega).$$

We deduce:

$$H_n^{\nu,0}(\gamma) = Y_n^{-\nu}(\gamma,0) = Y_n^\nu(\gamma,0).$$

□

Finally, with (A.3) and because the normalized spherical harmonics form an orthonormal basis, we have:

$$T_n^{-\nu,-m}(\gamma,\chi) = \overline{T_n^{\nu,m}(\gamma,\chi)}, \tag{3.14}$$

and therefore with (3.12):

$$H_n^{-\nu,-m}(\gamma) = H_n^{\nu,m}(\gamma). \tag{3.15}$$

Moreover (see (5.52) in [GD03]) we have also:

$$H_n^{\nu,m}(\gamma) = H_n^{m,\nu}(\gamma). \tag{3.16}$$

These two equations, (3.15) and (3.16), are useful to fasten the computation of the $H_n^{\nu,m}(\gamma)$ (see below).

Computation of $T_{O_j}^{\nu,k}$ and $T_{I_j}^{\nu,k}$. From the definition of the Outer and Inner functions, (1.1) and (1.2), and the definition of their corresponding rotation coefficients, (3.2) and (3.3), we deduce with (3.1) :

$$T_{O_j}^{\nu,k} = i^{|k|-|\nu|} \frac{A_j^\nu}{A_j^k} T_j^{\nu,k}, \tag{3.17}$$

and:

$$T_{I_j}^{\nu,k} = i^{|\nu|-|k|} \frac{A_j^k}{A_j^\nu} T_j^{\nu,k}. \tag{3.18}$$

With (3.14), we have also:

$$T_{O_j}^{-\nu,-k} = (-1)^{k+\nu} \overline{T_{O_j}^{\nu,k}},$$
$$T_{I_j}^{-\nu,-k} = (-1)^{k+\nu} \overline{T_{I_j}^{\nu,k}}.$$

Computation procedure. Given the $H_n^{\nu,0}(\gamma)$ with lemma 3.1 and (3.15) for $n = [\![0..P]\!]$ and $|\nu| \leq n$, we use (3.13) and (3.16) to compute the $H_n^{\nu,m}(\gamma)$ for $n = [\![0..P]\!]$, $|\nu| \leq n$ and $|m| \leq n$. $T_{O_j}^{\nu,k}$ and $T_{I_j}^{\nu,k}$ are then deduced from $H_n^{\nu,m}$ with (3.12), (3.17) and (3.18).

However there is no need for the computation of $T_{O_j}^{\nu,k}$ and $T_{I_j}^{\nu,k}$ to be accelerated: it will indeed be part of the precomputation of $M2L$ transfer function for a given level (see section 1.1) and its runtime is not critical at all. It is here more important to focus on the numerical stability of the computation scheme used for these rotation coefficients (see section 3.4).

3.1.3 Complexity

As for the operation count study of the FFT improvement for $M2L$, we here do not consider the precomputation performed once for the each level. The $T_{O_j^{\nu,k}}(\pi,\theta,\phi)$, $T_{I_j^{\nu,k}}(\phi,\theta,\pi)$ and $O_j^k(r,0,0)$ are therefore not taken into account.

For 1 $M2L$, we thus consider:

- 1 rotation for the multipole expansion
- 1 $M2L$ along the **z** axis
- 1 rotation for the local expansion

Operation count 3.1. *The theoretical operation count of one $M2L$ operation with rotations and single height kernel is:*

$$M2L_{ROT} = \frac{(P+2)(14P^2+41P+36)}{24}.$$

Proof. We have, with $Q = P/2$ (integer division):

$$ROT(Multipole) = \sum_{\nu=0}^{Q}\sum_{j=\nu}^{P-\nu}\sum_{k=-j}^{j} = \frac{(P+1)(P+2)^2}{4} \approx \frac{1}{4}P^3,$$

$$M2L_z = \sum_{j=0}^{P}\sum_{k=0}^{\min(j,P-j)}\sum_{n=k}^{P-j} = \frac{(2P+3)(P+4)(P+2)}{24} \approx \frac{1}{12}P^3,$$

$$ROT(Local) = \sum_{j=0}^{P}\sum_{\nu=0}^{j}\sum_{k=-Min(j,P-j)}^{Min(j,P-j)} = \frac{(P+2)(P^2+2P+2)}{4} \approx \frac{1}{4}P^3.$$

□

Operation count 3.2. *The theoretical operation count of one $M2L$ operation with rotations and double height kernel is:*

$$M2L_{ROT2P} = \frac{(10P+9)(P+2)(P+1)}{6}.$$

Proof. We have:

$$ROT(Multipole) = \sum_{\nu=0}^{P}\sum_{j=\nu}^{P}\sum_{k=-j}^{j} = \frac{(4P+3)(P+2)(P+1)}{6} \approx \frac{2}{3}P^3,$$

$$M2L_z = \sum_{j=0}^{P}\sum_{k=0}^{j}\sum_{n=k}^{P} = \frac{(2P+3)(P+2)(P+1)}{6} \approx \frac{1}{3}P^3,$$

$$ROT(Local) = \sum_{j=0}^{P}\sum_{\nu=0}^{j}\sum_{k=-j}^{j} = \frac{(4P+3)(P+2)(P+1)}{6} \approx \frac{2}{3}P^3.$$

□

The ratio between single and double height kernels with rotation is then:

$$\frac{M2L_{ROT2P}}{M2L_{ROT}} = \frac{4(10P+9)(P+1)}{14P^2+41P+36} \approx \frac{20}{7}.$$

This ratio favors the double height kernel compared to classic $M2L$ implementation (ratio of 6, see section 1.2.4) or block FFT (ratio of 8, see section 2.9).

3.2 Memory requirements

The implementation of the rotation scheme do not encounter specific problems except careful writing of the loops in the single height $M2L$ kernel case. The rotation coefficients are easily stored in three-dimensional arrays. We now detail the extra memory required.

For each $M2L$ preformed with rotations we need first two auxiliary arrays where are stored the multipole and local expansions along the $\mathbf{z}$ axis. These arrays have the same size of an expansion with double height kernel, and even less with single height: in both case their memory usage can be neglected. Instead of an expansion, up to P or $2P$, for the classic $M2L$ computation, we now need for each $M2L$ vector:

- the corresponding $M2L$ transfer function along the $\mathbf{z}$ axis containing only terms with null order: its size is therefore $P+1$ or $2P+1$ and each term is real number,
- the two arrays containing the corresponding $T_{O_j}^{\nu,k}$ and $T_{I_j}^{\nu,k}$ complex terms for $0 \leq j \leq P$, $|k| \leq j$ and $0 \leq \nu \leq j$. We indeed do not need terms with $-j \leq \nu < 0$. The size of each array is: $\sum_{n=0}^{P}\sum_{\nu=0}^{n}\sum_{k=-n}^{n} = \frac{(4P+3)(P+2)(P+1)}{6}$. This applies to both single and double height kernels.

Finally we thus have, for single height $M2L$ kernel:

$$Mem_{Sg}(P,H) = \mathcal{N}(H)(P+1)(P+2)c + \mathcal{N}_{\mathcal{T}}\left(\frac{P+1}{2} + \frac{(4P+3)(P+2)(P+1)}{3}\right)c, \tag{3.19}$$

and for double height kernel:

$$Mem_{Db}(P,H) = \mathcal{N}(H)(P+1)(P+2)c + \mathcal{N}_{\mathcal{T}}\left(\frac{2P+1}{2} + \frac{(4P+3)(P+2)(P+1)}{3}\right)c. \tag{3.20}$$

When compared to equations (1.14) and (1.15), it is clear that, unless very low octree heights and very high values of P, the extra memory storage needed for the rotation scheme is small.

3.3 BLAS usage study

We here consider the use of standard BLAS calls in order to speed up the computation of $M2L$ with rotations (see section 4 for an introduction on BLAS). We first study the double height kernel since this is the most amenable case.

3.3.1 Double height kernel

When using rotations with $M2L$, we use 2 different kinds of computation: rotations of multipole and local expansions on one side, and $M2L$ along the $\mathbf{z}$ axis on the other side.

Expansion rotations. We focus on the multipole expansion rotation: the local expansion rotation is performed likewise. We need to compute negative orders (see (3.6)) like:

$$\widehat{M}_n^{-\nu} = \sum_{l=-n}^{n} M_n^l T_{O_n}^{\nu,-l}(\omega,\gamma,\chi)$$

with: $0 \leq j \leq P$ and $0 \leq |\nu| \leq j$ for double height kernel. This computation does not lend itself to a matrix-vector product since it looks like:

$$c_{\nu,n} = \sum_{l=\dots}^{\dots} a_{n,\nu,l}.b_{l,n}$$

when we would need something like:

$$c_{\nu,n} = \sum_{l=\dots}^{\dots} a_{\nu,l}.b_{l,n}$$

for $\mathbf{C} = \mathbf{A}.\mathbf{B}$, with $\widehat{M}_n^{-\nu}$ stored in $\mathbf{C}$, $T_{O_n}^{\nu,-l}$ in $\mathbf{A}$ and M_n^l in $\mathbf{B}$. The dependency on n of $T_{Outer_n}^{\nu,-l}$ prevents us from considering the rotation of multipole expansion as a matrix-vector product, and thus from using level 2 BLAS, and furthermore level 3 BLAS.

It is however possible to use level 1 BLAS for the computation of one given $\widehat{M}_n^{-\nu}$, but the speedup with level 1 BLAS is limited.

$M2L$ **along z axis.** When considering (3.8), we can use the following matrix-vector product (represented for $P = 3$) for its computation:

$$\begin{bmatrix} \widehat{L}_0^0 & X & X & X \\ \widehat{L}_1^0 & \widehat{L}_1^1 & X & X \\ \widehat{L}_2^0 & \widehat{L}_2^1 & \widehat{L}_2^2 & X \\ \widehat{L}_3^0 & \widehat{L}_3^1 & \widehat{L}_3^2 & \widehat{L}_3^3 \end{bmatrix} = \begin{bmatrix} O_0^0 & O_1^0 & O_2^0 & O_3^0 \\ O_1^0 & O_2^0 & O_3^0 & O_4^0 \\ O_2^0 & O_3^0 & O_4^0 & O_5^0 \\ O_3^0 & O_4^0 & O_5^0 & O_6^0 \end{bmatrix} \begin{bmatrix} \widehat{M}_0^0 & 0 & 0 & 0 \\ \widehat{M}_1^0 & \widehat{M}_1^{-1} & 0 & 0 \\ \widehat{M}_2^0 & \widehat{M}_2^{-1} & \widehat{M}_2^{-2} & 0 \\ \widehat{M}_3^0 & \widehat{M}_3^{-1} & \widehat{M}_3^{-2} & \widehat{M}_3^{-3} \end{bmatrix}$$

But all the X terms in the local expansion matrix are uselessly computed and the 0 of the multipole expansion matrix are uselessly used. We can avoid the use of 0 with triangular BLAS (like $ZTRMM$, see [DCHD90]) with transposition:

$$\begin{bmatrix} \widehat{L}_0^0 & \widehat{L}_1^0 & \widehat{L}_2^0 & \widehat{L}_3^0 \\ X & \widehat{L}_1^1 & \widehat{L}_2^1 & \widehat{L}_3^1 \\ X & X & \widehat{L}_2^2 & \widehat{L}_3^2 \\ X & X & X & \widehat{L}_3^3 \end{bmatrix} = \begin{bmatrix} \widehat{M}_0^0 & \widehat{M}_1^0 & \widehat{M}_2^0 & \widehat{M}_3^0 \\ 0 & \widehat{M}_1^{-1} & \widehat{M}_2^{-1} & \widehat{M}_3^{-1} \\ 0 & 0 & \widehat{M}_2^{-2} & \widehat{M}_3^{-2} \\ 0 & 0 & 0 & \widehat{M}_3^{-3} \end{bmatrix} \begin{bmatrix} O_0^0 & O_1^0 & O_2^0 & O_3^0 \\ O_1^0 & O_2^0 & O_3^0 & O_4^0 \\ O_2^0 & O_3^0 & O_4^0 & O_5^0 \\ O_3^0 & O_4^0 & O_5^0 & O_6^0 \end{bmatrix} \tag{3.21}$$

The X terms in the local expansion matrix are however still uselessly computed. Moreover the speedup offered by the BLAS is at best with level 3: this would require the concatenation of several multipole/local expansion matrices in (3.21) as in section 4.3. But then the matrix containing the multipole expansions would not be triangular anymore: the use of level 3 BLAS is thus inappropriate. At last as shown in section 3.1.3, the $M2L$ along the **z** axis is the least time-consuming part in the $M2L$ computation with rotations. All these reasons lead us to foresee the gain in matching the $M2L$ computation with rotations with standard BLAS calls as too limited to justify its implementation.

The only way to speed up computations with rotations seems therefore to write special hand coded and non portable loops including assembly level programming that optimize computations in the same way BLAS implementations do.

3.3.2 Single height kernel

Since the computation for single height kernel does even less lend itself to a matrix-vector product (the triangular matrices in the double height case being further "halved" along the secondary (or skew) diagonal), the BLAS computation would be even less efficient.

We have therefore choosen not to use BLAS calls with rotations since we do not believe this would give sufficient gain.

3.4 Numerical stability

The numerical stability of the recursive computation of rotation coefficients has been studied for example in [CIGR99] and [WHG96] but for other recursive formulae than (3.13). To check our formulae we use numerical tests on the whole FMM computation: as shown in figure 3.2 no numerical instability is introduced with our rotation improvement since both curves are identical. This is also valid for double height $M2L$ kernel and for higher octree height.

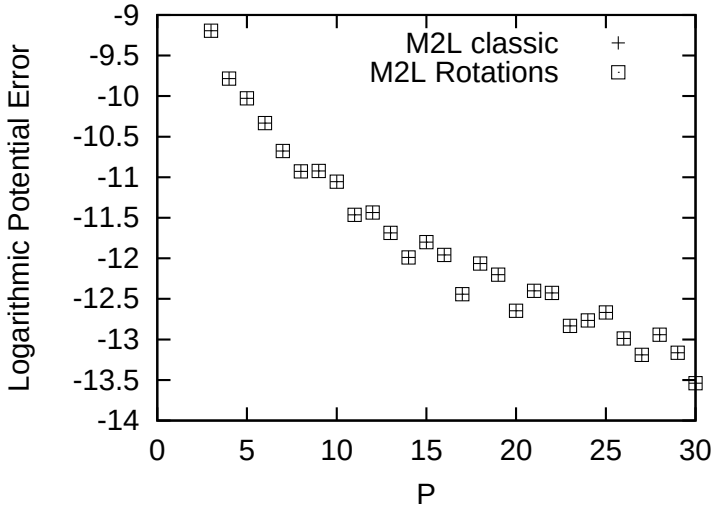


Figure 3.2: Logarithmic error for the potential according to P for single height $M2L$ kernel with and without rotations for $M2L$ computation. Tests performed on 10000 uniform distribution with an octree of height 4. The error plotted is the maximum absolute error over all the particles.

4 Implementation with BLAS

The BLAS (Basic Linear Algebra Subprograms) (see [LHKK79] [DCHH88] [DCHD90]) are a standard interface for some usual linear algebra operations such as a dot product of 2 vectors (level 1 BLAS), a matrix-vector product (level 2 BLAS) or a matrix-matrix product (level 3 BLAS). In order to offer an efficient implementation of these operations, the BLAS routines manage to fill at best the pipelines of the floating point units, thanks to an optimal use of the different layers of the hierarchical memory of the computer. And the higher the level of the BLAS used, the better the speedup they reach. The BLAS are moreover portable and widely available.

BLAS have already been used for hierarchical $\mathcal{O}(N)$ N-body algorithms by Hu and Johnsson [HJ96] with Anderson's method [And92] which uses different expansions than the FMM, and hence translation/conversion operators, but has the same algorithm for the upward and downward passes.

Here we propose the first BLAS implementation for the $M2L$ operator of the FMM for both single and double height kernel. In order to achieve the highest efficiency, we also detail several ways to use level 3 BLAS for $M2L$.

4.1 $M2L$ operator viewed as a matrix-vector product

While the original FMM formulae of Greengard & Rokhlin (see [GR88]) do not allow a rewriting of their corresponding operators ($M2M$, $M2L$ or $L2L$) as matrix-vector products, this can be easily done with the simpler formulae of [ED95], [WHG94] or [Ell95]. The full FMM algoritm has also been rewritten with matrices in [SP01] (with different notations): we here only focus on the rewriting of the $M2L$ operator as a matrix-vector product.

We first consider the double height $M2L$ kernel and contrary to [SP01], we only deal with non-null terms in multipole or local expansions, that is to say terms with degree j and order k such that $0 \leq j$ and $|k| \leq j$. We also point out that when performing $M2L$ operation, we compute only the terms with positive orders in the local expansion. A lot of symmetry relations in the transfer matrix (see its definition below) or in its sub-matrices could certainly be noticed when computing terms with negative orders too, but this would result in a slower computation than just computing positive order terms and then using (1.9) as we do. Moreover we now only consider linear algebra operations (such as those implemented in the BLAS): this is why we have to store terms with positive orders as well as terms with negative orders for the multipole expansions. The symmetry property (1.7) can indeed not be used here in order to write $M2L$ operator as a matrix-vector product while storing only the multipole expansion terms with positive orders.

Given a local expansion L_j^k for $0 \leq j \leq P$, $0 \leq k \leq j$, we define the corresponding *local expansion vector*, or *local vector*, of length $\frac{(P+1)(P+2)}{2}$ by:

$$\forall i_1 \in [\![0, \tfrac{(P+1)(P+2)}{2} - 1]\!], \quad \begin{cases} \mathbf{v}^L(i_1) = L_j^k, \\ i_1 = \frac{j(j+1)}{2} + k. \end{cases}$$

With $P = 2$, we have thus:

$$\mathbf{v}^L = \begin{bmatrix} L_0^0 \\ \hline L_1^0 \\ L_1^1 \\ \hline L_2^0 \\ L_2^1 \\ L_2^2 \end{bmatrix} .$$

Given a multipole expansion M_n^l for $0 \leq n \leq P$, $|l| \leq j$, we define the corresponding *multipole expansion vector*, or *multipole vector* of length $(P+1)^2$ by:

$$\forall i_2 \in [\![0, (P+1)^2 - 1]\!], \quad \begin{cases} \mathbf{v}^M(i_2) = M_n^l, \\ i_2 = n(n+1) + l. \end{cases}$$

With $P = 2$, we have thus :

$$\mathbf{v}^M = \left[\begin{array}{c} M_0^0 \\ \hline M_1^{-1} \\ M_1^0 \\ M_1^1 \\ \hline M_2^{-2} \\ M_2^{-1} \\ M_2^0 \\ M_2^1 \\ M_2^2 \end{array}\right] .$$

Given the $M2L$ transfer function, i.e. O_j^k with $0 \leq j \leq 2P$ and $|k| \leq j$, we define the corresponding $M2L$ *transfer matrix* of size $\frac{(P+1)(P+2)}{2} \times (P+1)^2$ by:

$$\forall (i_1, i_2) \in [\![0, \tfrac{(P+1)(P+2)}{2} - 1]\!] \times [\![0, (P+1)^2 - 1]\!], \quad \left\{ \begin{array}{l} \mathbf{T}_{M2L}(i_1, i_2) = O_{j+n}^{-k-l}, \\ i_1 = \frac{j(j+1)}{2} + k, \\ i_2 = n(n+1) + l. \end{array} \right.$$

With $P = 2$, we have thus :

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|ccc|ccccc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} \\ \hline O_2^0 & O_3^1 & O_3^0 & O_3^{-1} & O_4^2 & O_4^1 & O_4^0 & O_4^{-1} & O_4^{-2} \\ O_2^{-1} & O_3^0 & O_3^{-1} & O_3^{-2} & O_4^1 & O_4^0 & O_4^{-1} & O_4^{-2} & O_4^{-3} \\ O_2^{-2} & O_3^{-1} & O_3^{-2} & O_3^{-3} & O_4^0 & O_4^{-1} & O_4^{-2} & O_4^{-3} & O_4^{-4} \end{array}\right] . \tag{4.1}$$

The $M2L$ operator can now be computed as the following matrix-vector product:

$$\mathbf{v}^L = \mathbf{T}_{M2L} \cdot \mathbf{v}^M.$$

With single height $M2L$ kernel we have to consider the O_j^k in the $M2L$ *transfer matrix* that vanish for $j > P$, i.e. for $P = 2$:

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|ccc|ccccc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & 0 & 0 & 0 & 0 & 0 \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & 0 & 0 & 0 & 0 & 0 \\ \hline O_2^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] . \tag{4.2}$$

Remark 4.1. *We can of course also write $M2M$ and $L2L$ operators as matrix-vectors products.*

4.2 Level 2 BLAS

Now that we have presented $M2L$ operator as a matrix-vector product, it is simple to use level 2 BLAS calls to implement it: the speedup achieved by the BLAS is then directly exploitable.

Our matrices are stored in column storage mode, and we use the transfer matrix in its transposed form: it is indeed generally more efficient to compute $C \leftarrow A^T.B$ in row storage mode for A, than $C \leftarrow A.B$ with column storage mode for A, since when considering the BLAS implementation (see [BLA] for a default implementation) the first solution leads to less writings, with however more readings, than the second one.

4.2.1 Double height $M2L$ kernel

The matrix $\mathbf{T}_{M2L}$ is dense: the use of level 2 BLAS $ZGEMV$ routine is therefore obvious. In order to differentiate it from the other BLAS methods that will be used later, we name it *full_ blas*.

Remark 4.2. *Another possibility is to split* $\mathbf{T}_{M2L}$ *in a square matrix* $\mathbf{T'}_{M2L}$ *of size* $\frac{(P+1)(P+2)}{2} \times \frac{(P+1)(P+2)}{2}$ *that correspond to terms with positive (or null) order in the multipole expansion vector, and a matrix* $\mathbf{T''}_{M2L}$ *of size* $\frac{(P+1)(P+2)}{2} \times \frac{P(P+1)}{2}$ *that correspond to terms with strictly negative order in the multipole expansion. The interest here is that* $\mathbf{T'}_{M2L}$ *is a symmetric matrix, but the use of a "symmetric BLAS" such as* $ZSYMV$ *or* $ZSYMM$ *routines does not speed up the computation compared to the corresponding call to* $ZGEMV$ *or* $ZGEMM$ *routines: the main interest when using these symmetric BLAS is that only half of the matrix need to be written, which does not interest us here since the building of* $M2L$ *transfer matrices is precomputed and has thus very few impact on the CPU times. Tests done with level 3 BLAS (see section 4.3) and the corresponding block decomposition for* $\mathbf{T'}_{M2L}$ *and* $\mathbf{T''}_{M2L}$ *(see 4.3.3), have confirmed that such alternative does not improve the performances over the original ones.*

4.2.2 Single height $M2L$ kernel

The matrix $\mathbf{T}_{M2L}$ is now sparse: clearly the use of a level 2 BLAS call such as $ZGEMV$ routine would result in too many useless computations. We therefore have to split the sparse matrix-vector product in several dense block products.

Let's recall first techniques used in the BLAS implementation that lead to high performance (see [DDSvdV98]). The ultimate goal is here to fill at best the pipelines of the floating point execution units in order to reach peak performance of the processor. When writing loops corresponding to the BLAS operation, care is first taken in the order between the different loops so that spacial and temporal locality among data are maximized. Then, a first level of blocking is introduced to provide cache reuse in order to prevent multiple loads of the same data. When considering a multi-level cache memory and/or the TLB (Translation Lookaside Buffer), additional levels of blocking can be introduced. Unappropriate "leading dimensions" of the matrices (see BLAS routine interfaces) can cause unnecessary traffic in the memory hierarchy and an underused cache: local arrays are thus used to temporarily store sub-matrices. Other techniques such as loop unrolling, register blocking, data prefetch, etc. are also used depending on the machine. All these techniques are controlled by machine-dependent parameters.

In [KLvL98] it has been shown that level 3 BLAS routines can efficiently be implemented only with the $GEMM$ level 3 BLAS routine and a few level 2 BLAS routines. For portability purpose, as well as for simplicity, we prefer to adopt such approach. We will thus decompose the sparse matrix-vector product corresponding to $M2L$ operator in several $ZGEMV$ block products in the most efficient way. $ZGEMV$ routine is used here since we treat a matrix-vector product, but in section (4.3) we will see how to use matrix-matrix products, and $ZGEMM$ routine will then be used as in [KLvL98]. All the optimizations recalled above will be used but left as much as possible to the underlying (and machine dependent) $ZGEMV/ZGEMM$ BLAS routine called. We point out now one important fact: in our implementation of the FMM, we are free in the memory storage of the blocks of $\mathbf{T}_{M2L}$ since its construction is precomputed at each level of the octree in the downward pass of the FMM. Therefore, most of problems that arise when considering the leading dimension of a matrix will be irrelevant with our blocks. Moreover, we also recall that the values of P are limited: in practice they often range from 3 for low precision to 15 for high precision. We can therefore not include too many additional zeros in the block decomposition of the $M2L$ transfer matrix with single height kernel since every additional cost is amplified by the very high number of uses of the $M2L$ operator and especially costly for low values of P.

Before presenting the three block decompositions used, we first expose the underlying structure of $\mathbf{T}_{M2L}$. As suggested in its representation given in (4.1), this matrix can in fact be viewed as a "concatenation" of several sub-matrices denoted as $\mathbf{B}_{I_1,I_2}$.

Block structure of $\mathbf{T}_{M2L}$. For all $(I_1, I_2) \in [\![0, P]\!]^2$, let $\mathbf{B}_{I_1,I_2}$ denotes a matrix of size $(I_1+1) \times (2I_2+1)$ defined by:

$$\mathbf{B}_{I_1,I_2}(i_1, i_2) = O_{I_1+I_2}^{-i_1+I_2-i_2} \qquad \forall (i_1, i_2) \in [\![0, I_1]\!] \times [\![0, 2I_2]\!].$$

For a double height kernel the $\mathbf{T}_{M2L}$ matrix would have the following block structure:

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|c|c|c} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} & \dots & \mathbf{B}_{0,P} \\ \hline \mathbf{B}_{1,0} & \mathbf{B}_{1,1} & \dots & \mathbf{B}_{1,P} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{B}_{P,0} & \mathbf{B}_{P,1} & \dots & \mathbf{B}_{P,P} \end{array}\right].$$

Since we deal here with single height kernel, the block $\mathbf{B}_{I_1,I_2}$ is considered to vanish for $I_1 + I_2 > P$.
For $P = 2$ we have thus:

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|c|c} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} & \mathbf{B}_{0,2} \\ \hline \mathbf{B}_{1,0} & \mathbf{B}_{1,1} & 0_{3\times 5} \\ \hline \mathbf{B}_{2,0} & 0_{5\times 3} & 0_{5\times 5} \end{array}\right] \tag{4.3}$$

where $0_{M\times N}$ denotes the null matrix of size $M \times N$.

It can be noticed that when considering a block decomposition of the multipole vector too, we obtain a simple block product version of $M2L$ operator for single height kernel. Indeed when denoting by $\mathbf{v}_I^M$ for all $I \in [\![0, P]\!]$ the following vector of size $2I + 1$:

$$\mathbf{v}_I^M(i) = M_I^{i-I} \qquad \forall i \in [\![0, 2I]\!],$$

we have (for $P = 2$):

$$\mathbf{v}^M = \left[\begin{array}{c} \mathbf{v}_0^M \\ \hline \mathbf{v}_1^M \\ \hline \mathbf{v}_2^M \end{array}\right]$$

and:

$$\mathbf{T}_{M2L} \ . \ \mathbf{v}^M = \left[\begin{array}{c} \mathbf{B}_{0,0}\mathbf{v}_0^M + \mathbf{B}_{0,1}\mathbf{v}_1^M + \mathbf{B}_{0,2}\mathbf{v}_2^M \\ \hline \mathbf{B}_{1,0}\mathbf{v}_0^M + \mathbf{B}_{1,1}\mathbf{v}_1^M \\ \hline \mathbf{B}_{2,0}\mathbf{v}_0^M \end{array}\right] \quad .$$

Because of the greatly different sizes of the blocks $\mathbf{B}_{I_1,I_2}$, this simple block decomposition, with one BLAS call ($ZGEMV$ routine) for each product between 2 blocks, would however not lead to the best efficiency. Therefore we present now 3 possible decompositions that can be efficiently combined with BLAS calls.

band_blas decomposition. Our first decomposition is strictly based on the underlying structure of $\mathbf{T}_{M2L}$ as exposed in (4.3): this structure looks like upside-down stairs, each step having a different size (height and depth). For each step corresponds one single *band*.

The first choice to consider is to split $\mathbf{T}_{M2L}$ in *bands* either in the row (or horizontal) direction or in the column (or vertical) direction. Our matrices are stored by columns for BLAS calls, but since we use the transposed transfer matrix (see introduction of section 4.2), this one is stored by rows. Thus it seems at first better to consider *bands* in the row direction. Moreover, *bands* in the row direction imposes several traversals of the multipole expansion vector (one for each *band*) against one single traversal of the local expansion vector. In other words, with *bands* in the row direction, the resulting blocks of the local vector are updated one after the other, whereas the corresponding data of the multipole vector may be reloaded several times during the block matrix-vector product. On the other hand, *bands* in the column direction imposes several traversals of the local expansion vector against a single one on the multipole expansion vector. Since the local expansion vector is traversed for updating (i.e. both reading and writing) and the multipole expansion vector for reading, it is more efficient to use *bands* in the row direction.

There is $P + 1$ *bands* and the first one is thus the concatenation of blocks $\mathbf{B}_{0,0}$, $\mathbf{B}_{0,1}$ ... $\mathbf{B}_{0,P}$, the second one the concatenation of blocks $\mathbf{B}_{1,0}$, $\mathbf{B}_{1,1}$ ... $\mathbf{B}_{1,P-1}$, and so on until the last *band* which corresponds to block $\mathbf{B}_{P,0}$ only (see figure 4.1).

Each *band* is then stored separately and treated with one call to $ZGEMV$ routine. We store the *band* by rows and use its transpose for the BLAS call. The leading dimension is here the height of the *band*. It can be noted that the first *band* leads to a simple dot product while the last *band* leads to a point-wise product between 2 vectors: both do not offer substantial speedup.

One obvious problem with such decomposition is that, as P grows, the first *bands* are too thin and too long while the last ones are too thick and too short. This prevents the BLAS routine to fully decompose internally the given *band* according to the hierarchical memory of the computer. We will thus try to use *bands* with higher "height" (i.e. greater number of rows) in order to relax these constraints on the BLAS efficiency.

cband_blas decomposition. In [KLvL98], triangular matrix-matrix product was performed with $ZGEMM$ routines thanks to decomposition of the triangular matrix in "strips". These strips could be vertically or horizontally oriented leading to respectively block columns or block rows. The choice between these 2 orientations must favor the one that minimizes the number of updatings (i.e. both writings and readings) of data against the number of readings. As mentioned page 39 this is realized in our case with strips in the horizontal direction (i.e. block rows). We are thus left with a decomposition of $\mathbf{T}_{M2L}$ in horizontal strips whose "height" is customizable

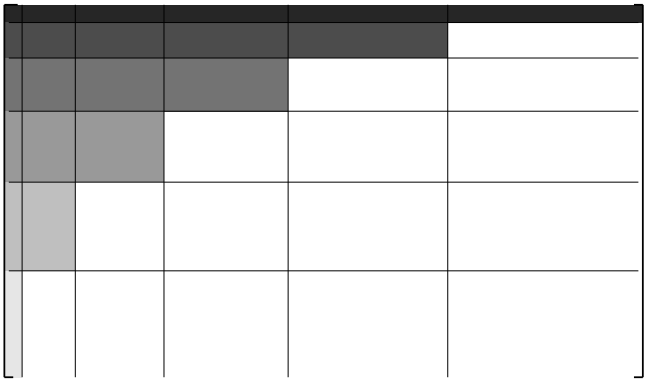


Figure 4.1: *band_ blas* decomposition for $P = 5$: the different gray values show the different *bands*.

but constant among all strips. We name such strip *cband* for "constant *band*" in reference to the *bands* used in the *band_ blas* decomposition, and their common height is hereafter named dim_{cband}.

When a *cband* covers, fully or partly, 2 consecutive *bands* (as defined for *band_ blas* decomposition), zeros have to be added at the end of the lower *band* so that the width of the *cband* equals the width of the upper *band*. We have thus implemented a better *cband_ blas* version, as confirmed by performance tests, that skip these zeros by using two BLAS calls: one for the *cband* with width equal to the lower *band* and another one for the remaining data at the end of the upper *band*. When a *cband* covers 3 or more *bands*, only the zeros between the first and the second *band* are skipped: this eliminates most of zeros while avoiding numerous BLAS calls with few operations.

The upper-right block of $\mathbf{T}_{M2L}$, namely the block $\mathbf{B}_{0,P}$ whose height is 1, and the lower-left block $\mathbf{B}_{P,0}$, whose width is 1, are treated separately: they would indeed supply too many zeros otherwise.

This is represented in figure 4.2.

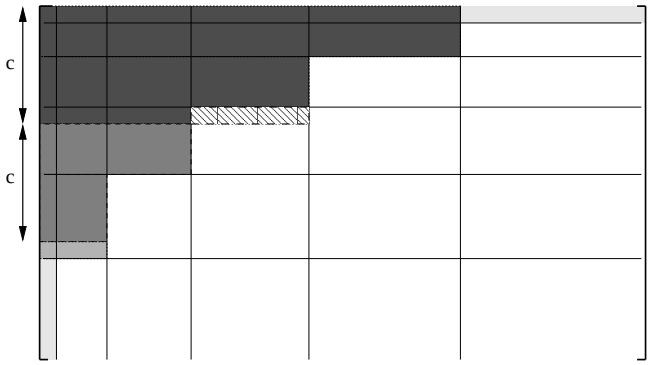


Figure 4.2: *cband_ blas* decomposition for $P = 5$: the different gray values show the different *cbands*. c denotes the dim_{cband} height and the additional zeros are pictured with stripes. The upper-right block and the lower-left block are treated separately.

As for *band_ blas* decomposition, *cbands* are stored separately, with a leading dimension equal to their constant height, and treated as transposed for $ZGEMV$ routine.

One drawback of *cband_ blas* decomposition is that with high dim_{cband} we introduce some useless zeros in the computation, while with small dim_{cband} we have thin and long rectangular blocks. That's why, depending on P and on the machine used, the right value for the height of our *cbands* have to be obtained experimentally.

block_ blas decomposition. Our last decomposition is based on the underlying structure of $\mathbf{T}_{M2L}$ with single height kernel imposed by $O_j^k = 0$, $\forall j > O$. This structure can in fact be considered as recursive. More precisely, we now denote $\mathbf{T}_{M2L}$ as $\mathbf{T}_{0,0}[P]$ where $\mathbf{T}_{I_1,I_2}[P]$, $0 \leq I_1 \leq P$, $0 \leq I_2 \leq P$, denotes the matrix of size

$\frac{(P+1)(P+2)}{2} \times (P+1)^2$ defined by:

$$\mathbf{T}_{I_1,I_2}[P] = \left[\begin{array}{c|c|c|c|c} \mathbf{B}_{I_1,I_2} & \mathbf{B}_{I_1,I_2+1} & \cdots & \mathbf{B}_{I_1,P-(I_1+1)} & \mathbf{B}_{I_1,P-I_1} \\ \hline \mathbf{B}_{I_1+1,I_2} & \mathbf{B}_{I_1+1,I_2+1} & \cdots & \mathbf{B}_{I_1+1,P-(I_1+1)} & 0 \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{B}_{P-(I_2+1),I_2} & \mathbf{B}_{P-(I_2+1),I_2+1} & \cdots & 0 & 0 \\ \hline \mathbf{B}_{P-I_2,I_2} & 0 & \cdots & 0 & 0 \end{array}\right] .$$

When $I_1 + I_2 \leq P$ this matrix is dense: we therefore denote it by $D_{I_1,I_2}[P]$. More precisely:

$$\mathbf{D}_{I_1,I_2}[P] = \left[\begin{array}{c|c|c|c} \mathbf{B}_{I_1,I_2} & \mathbf{B}_{I_1,I_2+1} & \cdots & \mathbf{B}_{I_1,P} \\ \hline \mathbf{B}_{I_1+1,I_2} & \mathbf{B}_{I_1+1,I_2+1} & \cdots & \mathbf{B}_{I_1+1,P} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{B}_{P,I_2} & \mathbf{B}_{P,I_2+1} & \cdots & \mathbf{B}_{P,P} \end{array}\right] .$$

We have now the following recursive decomposition for $0 \leq I_1 \leq P$, $0 \leq I_2 \leq P$:

- for odd values of $P = 2Q + 1$:

$$\mathbf{T}_{I_1,I_2}[P] = \mathbf{T}_{I_1,I_2}[2Q+1] = \left[\begin{array}{c|c} \mathbf{D}_{I_1,I_2}[Q] & \mathbf{T}_{I_1,I_2+Q+1}[Q] \\ \hline \mathbf{T}_{I_1+Q+1,I_2}[Q] & 0 \end{array}\right] , \tag{4.4}$$

- for even values of $P = 2Q$:

$$\mathbf{T}_{I_1,I_2}[P] = \mathbf{T}_{I_1,I_2}[2Q] = \left[\begin{array}{c|c} \mathbf{D}_{I_1,I_2}[Q] & \mathbf{T}_{I_1,I_2+Q+1}[Q-1] \\ \hline \mathbf{T}_{I_1+Q+1,I_2}[Q-1] & 0 \end{array}\right] .$$

We can then further decompose $\mathbf{T}_{I_1,I_2+Q+1}[\ldots]$ and $\mathbf{T}_{I_1+Q+1,I_2}[\ldots]$, which are sparse matrices too, until a terminal case determined by the value of Q.

For example, let's consider $P = 3$, we have:

$$\mathbf{T}_{M2L} = \mathbf{T}_{0,0}[3] = \mathbf{T}_{0,0}[2 \times 1 + 1] =$$

$$\left[\begin{array}{c|ccc|ccccc|ccccccc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^3 & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline O_2^0 & O_3^1 & O_3^0 & O_3^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-1} & O_3^0 & O_3^{-1} & O_3^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-2} & O_3^{-1} & O_3^{-2} & O_3^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline O_3^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_3^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_3^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_3^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

which can be decomposed following (4.4) in $\mathbf{D}_{0,0}[1]$:

$$\mathbf{D}_{0,0}[1] = \left[\begin{array}{c|cc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} \end{array}\right] ,$$

and in $\mathbf{T}_{0,2}[1]$ and $\mathbf{T}_{2,0}[1]$:

$$\mathbf{T}_{0,2}[1] = \left[\begin{array}{ccccc|ccccccc} O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^3 & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} \\ \hline O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$\mathbf{T}_{2,0}[1] = \left[\begin{array}{c|ccc} O_2^0 & O_3^1 & O_3^0 & O_3^{-1} \\ O_2^{-1} & O_3^0 & O_3^{-1} & O_3^{-2} \\ O_2^{-2} & O_3^{-1} & O_3^{-2} & O_3^{-3} \\ \hline O_3^0 & 0 & 0 & 0 \\ O_3^{-1} & 0 & 0 & 0 \\ O_3^{-2} & 0 & 0 & 0 \\ O_3^{-3} & 0 & 0 & 0 \end{array}\right] .$$

The advantages of this decomposition is that we use as much as possible dense matrices (i.e. $\mathbf{D}_{I_1,I_2}$) which are efficiently treated by $ZGEMV$ routine. We do not limit the BLAS efficiency by the height of the *band* as in the *band_blas* decomposition, and we do not compute useless zeros as in the *cband_blas* one. The terminal case are then treated as in *band_blas*.

It has to be noted than the recursive call on the sub-matrix "on the right" is performed before the recursive call on the sub-matrix "below": this leads indeed to less traversals on the local vector (for reading and writing) than on the multipole vector (for reading) as in page 39.

4.3 Level 3 BLAS

A matrix-vector product requires $\mathcal{O}(N^2)$ memory storage (supposing a square matrix) relative to $\mathcal{O}(N^2)$ operation count. Since the computation speed of modern processors is much faster than their memory bandwidth, it is generally not possible to continuously supply the floating point execution units with the needed data: these execution units remain then regularly idle waiting for data to be loaded from low level caches or main memory. Performance of level 2 BLAS is therefore limited by the rate of data movement through the hierarchical memory, and level 2 BLAS do not generally reach peak performance. On the contrary, a matrix-matrix product requires $\mathcal{O}(N^2)$ memory storage (supposing the matrices are square) relative to $\mathcal{O}(N^3)$ operation count. It is now easier to overlap memory latency with computation of the floating point execution units, and thus to reach peak performance of the processor. More precisely a blocking level is introduced in the loops that computes the matrix-matrix product $C \leftarrow A^T.B$, which corresponds to a block version of the product: all computations involving one given block of A are thus performed before loading the new block of A. This cache reuse prevents multipole loads for data of A that would arise in the non-block version of the product. We will therefore try to group multiple $M2L$ operations in one single operation that would correspond to a matrix-matrix product.

During the downward pass, at a given level of the octree, all $M2L$ operations that have the same $M2L$ vector between the center of the source cell (containing the multipole expansion) and the center of the target cell (containing the local expansion) share the same Outer functions (i.e. O_j^k) and thus the same $M2L$ transfer matrix. When considering all pairs of multipole / local expansions that share the same $M2L$ vector, it is thus possible to concatenate all their multipole expansion vectors as columns of one single *multipole expansion matrix* $\mathbf{M}^M$, also named *multipole matrix*. The local expansion vectors are also concatenated according to the same order to form one single *local expansion matrix* $\mathbf{M}^L$, also named *local matrix*. The matrix-matrix product $\mathbf{M}^L = \mathbf{T}_{M2L} \,.\, \mathbf{M}^M$ computes then the corresponding $M2L$ operations all at once as described in figure 4.3. It has to be noticed that the column storage of our matrices for the BLAS calls is clearly adapted for the concatenation of vectors as matrix columns.

First, we will roughly consider that the best efficiency is obtained with level 3 BLAS when the maximum number of multipole / local expansions are concatenated each time (see section 4.3.3 for revisions of this assertion). In the following, we will thus see how to concatenate the maximum number of multipole / local expansions for one given $M2L$ transfer matrix, and we will then present the implementation of this matrix-matrix product thanks to level 3 BLAS calls.

When grouping expansions according to $M2L$ vectors of the interaction list, we also have to take account that some cells do not have a complete interaction list: with free-space boundary conditions (FBC), where all the space outside of the computational box is considered as empty, this concerns the "border cells" in each level, that is to say the cells located at the boundaries of the computational box. In order to ease their computation, and for efficiency purpose, these cells with incomplete interaction list are computed separately when using level 3 BLAS for $M2L$ computation. In the case of Periodic Boundary Conditions (PBC), where the computational box is periodically replicated in each dimension, all cells , at each level, have a complete interaction list and such differentiation is irrelevant.

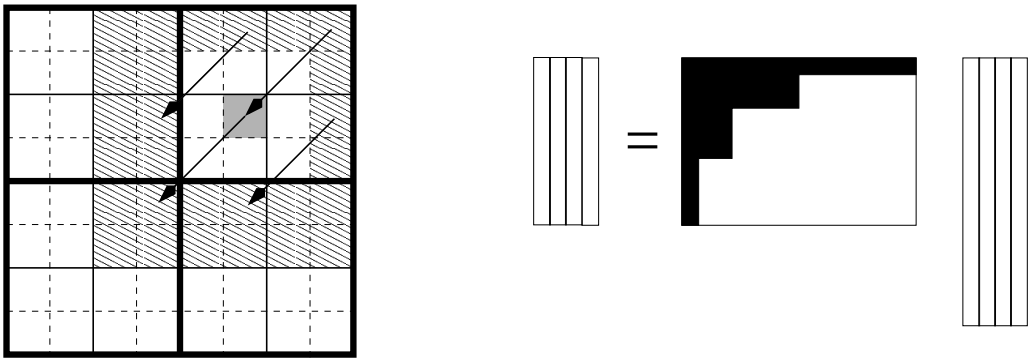


Figure 4.3: Concatenation of multipole and local expansions in order to use level 3 BLAS. The interaction list of the cell in gray is marked with stripes. Each of its $M2L$ operations is computed here with 3 other $M2L$ operations, that share the same $M2L$ vector, in one single matrix-matrix product: the corresponding multipole and local vectors are therefore concatenated in one multipole matrix and in one local matrix, both with 4 columns. The transfer matrix represented here is for a single height kernel.

4.3.1 Computing the local expansions of cells with incomplete interaction list

With free-space boundary conditions, the "cells with incomplete interaction list" are all the cells whose parent have at least one neighbor that is outside of the octree. See for example figure 4.4.

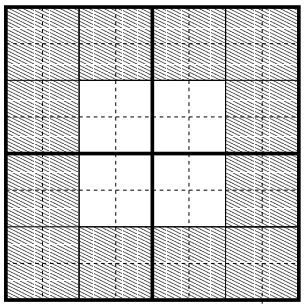


Figure 4.4: Cells with incomplete interaction list (with stripes) for a quadtree of height 3 and a well-separateness criterion $ws = 1$.

In order to treat all the $M2L$ operations for the local expansions of such cells, we use *recopies* of the multipole and local expansion vectors. Concretely, the current local expansions are copied in the columns of the local matrix, the corresponding multipole expansions are copied in the columns of the multipole matrix, and the $M2L$ computation is performed as: $\mathbf{M}^L \leftarrow \mathbf{T}_{M2L} \,.\, \mathbf{M}^M + \mathbf{M}^L$. The columns of the local matrix are then *recopied* in the original local expansions. It could have been possible to use a null local matrix and to add the result to the local expansions, but on modern processor such as PowerPC or IA-64 (Itanium) the fused multiply-add instruction (FMA, or FMADD) allows computation of $c = a.b + c$ in the same number of cycles as $c = a.b$. We therefore prefer 2 copies against 1 filling with zeros plus 1 addition.

For a given well-separateness criterion ws (see section 1.1), the $M2L$ vectors of the interaction list of a given cell are determined according to the *type of child* of the cell: *type of child* describes here the location of the cell center relatively to the center of its father. In $3D$, the eight different *type of child* are: BDL, BUL, BDR, BUR, FDL, FUL, FDR, FUR, with F, B, U, D, L, R corresponding respectively to "Front", "Back", "Up", "Down", "Left" and "Right". Two different *types of child* share some $M2L$ vectors. Some $M2L$ vectors are also shared by all the *types of child*: for example, with $ws = 1$, among the 189 $M2L$ vectors of the interaction list $5^3 - 3^3 = 98$ are common to all *types of child*.

When grouping $M2L$ operations according to the $M2L$ vector, we treat $M2L$ vectors one after the other. Two loops are thus needed: one for the *type of child* and another one for the $M2L$ vector. $\mathcal{V}_{M2L}$ denoting the set of all possible $M2L$ vectors, $\mathcal{T}_C$ the set of all different *types of child* and $\mathbf{T}_{M2L}(v)$ the $M2L$ transfer matrix corresponding to the $M2L$ vector v, the first possibility is:

```
- ∀ v ∈ V_M2L do:
   - ∀ t ∈ T_C corresponding to v do:
      - copy the local vectors in M^L ;
      - copy the corresponding multipole vectors in M^M ;
     end do
   - perform M^L = T_M2L . M^M ;
   - copy back all the local vectors copied in the loop on t ;
  end do.
```

And the second possibility is:

```
- ∀ t ∈ T_C do:
   - ∀ v ∈ V_M2L corresponding to t do:
      - copy the local vectors in M^L ;
      - copy the corresponding multipole vectors in M^M ;
      - perform M^L = T_M2L . M^M ;
      - copy back the corresponding local vectors;
     end do
  end do.
```

The number of *recopies* is the same in both possibilities, but in the first one the number of vectors treated per matrix-matrix product is higher than in the second one. Indeed in the first algorithm, for each $M2L$ vector there is only one level 3 BLAS call that computes all the local vectors concerned by this $M2L$ vector, no matter the *type of child* of the cell they belong to. Whereas in the second algorithm, for one $M2L$ vector there is 8 level 3 BLAS calls: one for each *type of child*. Since the speedup obtained with the use of level 3 BLAS increases with the number of vectors treated, we prefer to use the first algorithm. Moreover the first algorithm allows the reuse of the same memory area to successively store all different $M2L$ transfer matrices: there is no need to keep at the same time in memory all the $M2L$ transfer matrices.

It can be noted that the same algorithm could be used for an adaptive version of the FMM [CGR88].

4.3.2 Computing the local expansions of cells with complete interaction list

All cells with complete interaction list have the same interaction list size, and all cells of the same *type of child* share exactly the same $M2L$ vectors. This regularity allows well suited algorithms. We thus study 2 different methods to concatenate in practice our multipole / local expansion vectors:

- a simple one that always uses *recopies* in order to treat altogether the maximum number of pairs of multipole/local expansions;
- another one that avoids the additional cost of *recopies* thanks to a special *data storage* of our multipole and local vectors. This implies a "rearrangement" of the storage in memory of these vectors: this is done before the downward pass. The multipole and local vectors can be stored row by row, slice by slice or even for the whole level, but it implies (except for *row* storage) the useless computation of some local expansions belonging to what we will name *blank boxes*.

It has to be noted that for levels lower or equal to 2 there is no cells with complete interaction list.

Scheme with recopies. As in [HJ96], we consider one $M2L$ vector after the other, and for each we copy all corresponding pairs of multipole / local vectors in the multipole and local matrices and then perform one single matrix-matrix product. There is three possibilities:

1. ```
   - ∀ t ∈ T_C do:
      - copy all local vectors in M^L ;
      - ∀ v ∈ V_M2L corresponding to t do:
         - copy the corresponding multipole vectors in M^M ;
         - perform M^L = T_M2L . M^M ;
        end do
      - copy back all the local vectors ;
     end do.
   ```
2. ```
   - ∀ v ∈ V_M2L do:
      - ∀ t ∈ T_C corresponding to  v do:
         - copy all local vectors in M^L ;
   ```

```
        - copy the corresponding multipole vectors in M^M ;
        - perform M^L = T_M2L . M^M ;
        - copy back all local vectors ;
       end do
    end do.
3. - copy all local vectors in M^L ;
   - ∀ v ∈ V_M2L common to all types of child, do:
     - copy the corresponding multipole vectors in M^M ;
     - perform M^L = T_M2L . M^M ;
    end do
   - copy back all the local vectors;
   - ∀ t ∈ T_C do:
     - copy all local vectors in M^L ;
     - ∀ v ∈ V_M2L corresponding to t
       (and non-common to all types of child), do:
       - copy the corresponding multipole vectors in M^M ;
       - perform M^L = T_M2L . M^M ;
       end do
     - copy back all local vectors ;
    end do.
```

For the last algorithm, we recall that, among the 189 $M2L$ vectors of the interaction list, 98 are common to all *types of child*. The 91 remaining are used in the second loop (on t) of this last algorithm.

The second algorithm, compared to the first one, has the drawback to perform more copies for the local vectors while having the same number of vectors treated each time. The third algorithm, compared to the first one, presents the advantage to treat 8 times more vectors for 98 $M2L$ vectors out of 189. But it has the drawback to perform twice more copies for the local vectors than the first algorithm. Moreover, we believe that the number of vectors treated in the first algorithm is already sufficient to gain enough efficiency with the level 3 BLAS. Indeed for a level l with FBC, there is, in each dimension, $2^{l-1} - 2ws$ cells of the same *type of child* and with complete interaction list. $(2^{l-1} - 2ws)^3$ vectors are thus treated all at once with the first algorithm, which is 2744 for level 5, and is therefore high enough for most values of P. That is why we choose the first algorithm.

However one drawback with the first algorithm is the need to keep in memory all the $M2L$ transfer matrices during the $M2L$ computation of a level.

Row data storage. The first possibility in order to avoid *recopies* is to choose one given dimension and to store consecutively in memory all the local vectors that belong to cells of the same *type of child* along a given row in this dimension (see figure 4.5). This is done for each row of the level, and the same storage is also done for the multipole vectors. Note however that for local expansion vectors, only cells with complete interaction list have their local expansions rearranged in rows, while for the multipole expansions the rearrangement is performed for all cells of the level: this is because local expansions of cells with complete interaction list may require multipole expansions of cells with incomplete interaction list.

With such data storage we can call a level 3 BLAS routine starting at the local vector of the first cell of each row, with the corresponding $M2L$ transfer matrix and the corresponding multipole vector. Since the local vector of the next cell in the row, which has same *type of child*, is consecutive in memory and because it is the same for the multipole vectors, we can here use level 3 BLAS with local and multipole matrices stored by columns.

With FBC, at level l, there is 8^{l-1} cells of a given *type of child*. For each *type of child*, the multipole expansion vectors are rearranged in $(2^{l-1})^2$ rows of size 2^{l-1}. There is $(2^{l-1} - 2.ws)^3$ cells of a given *type of child* with complete interaction list: the local expansions are thus rearranged in $(2^{l-1} - 2.ws)^2$ rows of size $2^{l-1} - 2.ws$. The number of local expansions treated in each matrix-matrix product is therefore: $2^{l-1} - 2.ws$.

Remark 4.3. *In order to implement the row storage, we have to be able to identify (and access to the content of) any cell thanks to the coordinates of its center. This is enabled in our implementation thanks to the Morton ordering (see section 1.1) and this is also required by slice and level storages.*

Slice data storage. The main drawback of the *row* data storage is that the number of local vectors computed each time might be quite small for the first levels of the octree: at level $l = 4$, this equals 6 which is not enough to obtain the best efficiency with the level 3 BLAS.

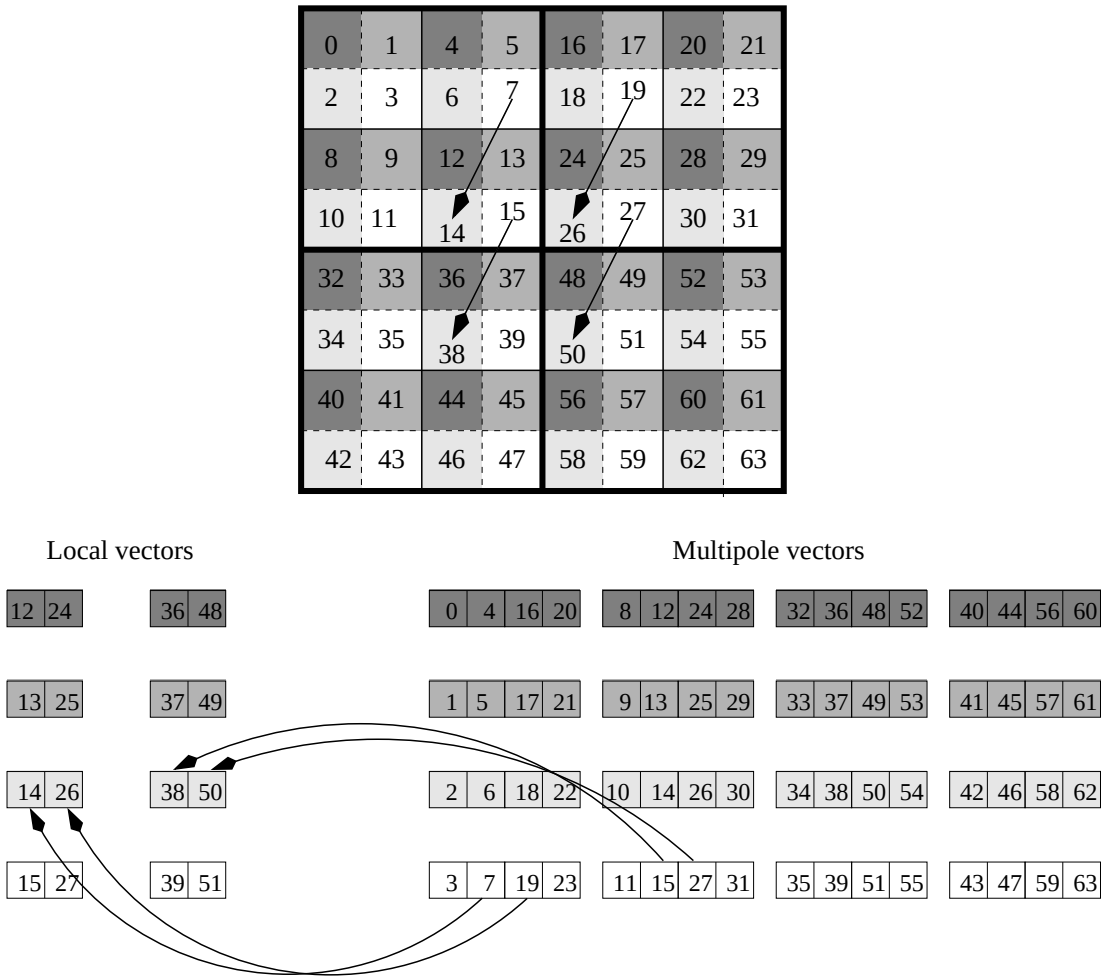


Figure 4.5: *row* data storage ($2D$, $ws = 1$, level 3). The cells are indexed according to Morton ordering. Each of the 4 *types of child* has a different gray value. The expansions of the cells with same *type of child* along a given row are concatenated: the four $M2L$ operations, whose $M2L$ vectors are represented on the quadtree, can then be directly computed with matrix-matrix products (level 3 BLAS) without *recopies*.

That's why we propose here to store consecutively the rows in memory in order to form a "slice" and then to treat several rows with one level 3 BLAS call. In order to have a correct correspondence between slices of local expansions and slices of multipole expansions, we have however to insert "blank cells" (or *blank boxes*) between rows of local expansions: these correspond to cells not belonging to the octree whose local vectors are computed uselessly. With $ws = 1$, there is 1 *blank box* at the beginning and 1 at the end of each row (see figure 4.6). We can however skip the first *blank box* of the slice (that is to say, the first *blank box* of the first row), and also the last one.

As for *row* storage, rearrangement of local expansion vectors only concerns cells with complete interaction list while rearrangement of multipole expansion vectors concerns all cells. Moreover, there is only *blank boxes* between rows of concatenated local expansion vectors. There is no need for such *blank boxes* for the rows of multipole expansion vectors: we just have to concatenate all multipole expansions of the whole slice in order to have a correct correspondence between the slice formed by the local expansions and the *blank boxes*, and the slice formed by the multipole expansions.

With FBC, at level l, and for each *type of child*, the multipole expansion vectors are rearranged in 2^{l-1} slices of size $(2^{l-1})^2$. The local expansion vectors are rearranged in $2^{l-1} - 2.ws$ slices of size $(2^{l-1} - 2.ws)^2 + 2 * (2^{l-1} - 2.ws) - 2$ (i.e. the number of cells of a given *type of child* with complete interaction list in 1 slice, plus 1 *blank box* at the end and at the beginning of each row, minus the first and the last *blank box* of the slice), which equals the number of local expansions treated in each matrix-matrix product.

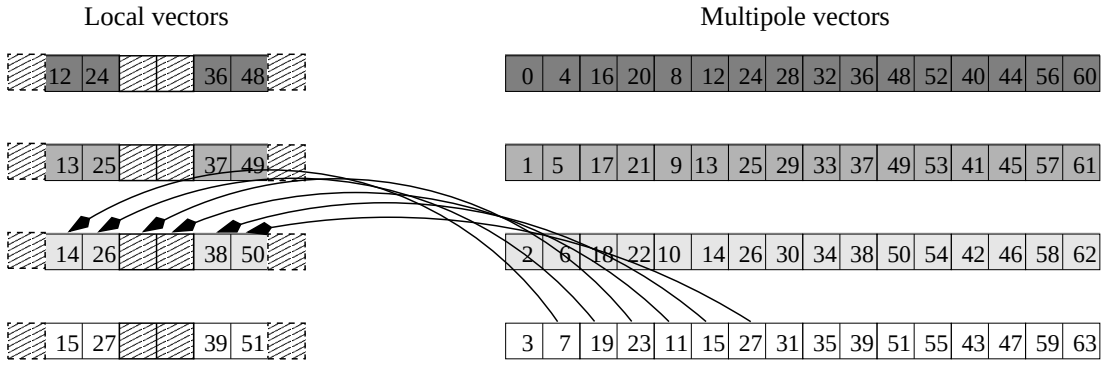


Figure 4.6: *slice* data storage ($2D$, $ws = 1$, level 3): see also figure 4.5. Each *type of child* has a different gray value. The rows of the *row* data storage mode are concatenated with *blank boxes* (marked with stripes) in between: the matrix-matrix product now involves multipole and local matrices with 6 columns, against 2 columns with *row* storage (see figure 4.5).

If *slice* data storage allows a better efficiency with the level 3 BLAS, its clear drawback is that it also performs useless extra work due to the *blank boxes*.

Level data storage. In the same way that we have concatenated rows to form slices in order to obtain better level 3 BLAS efficiency, we can concatenate slices in order to have the whole level stored in one single memory block and computed with one single level 3 BLAS call. This is called *level* data storage.

In addition to the *blank box* at the beginning and at the end of each row, we have to add for *level* storage a row of *blank boxes* between each slice of local vectors (no need for the multipole vectors). The first row of *blank boxes* of the level, as well as the last one, can be skipped.

The level 3 BLAS efficiency is higher, especially for low levels, but the number of *blank boxes* uselessly computed is clearly increased too. This has not yet been implemented.

4.3.3 Implementation with level 3 BLAS calls

As in section 4.2, the matrix-matrix product for double height $M2L$ kernel can be directly implemented with one single BLAS call, while for single height $M2L$ kernel the sparse transfer matrix has to be decomposed in blocks. The discussion of section 4.2 is here also valid when replacing $ZGEMV$ routine by $ZGEMM$ one.

However, with single height $M2L$ kernel, we have to face a new constraint when decomposing the matrix-matrix product in several level 3 BLAS calls. Indeed we have assumed at the beginning of this section that the more the number of columns in the multipole / local matrices, the best the efficiency we obtain for the corresponding level 3 BLAS call. If only one single level 3 BLAS call is performed for the matrix-matrix product, as for double height $M2L$ kernel (see section (4.2.1)), this is mainly true: tests have confirmed that even if some higher performance can be obtained when splitting the local and multipole matrices (as below), the gain in performance is relatively small since the single level 3 BLAS call performs its own splitting.

But with single height kernel, we have several level 3 BLAS calls per matrix-matrix product. If all the columns of the multipole or local matrix cannot be stored in a level of the hierarchical memory (level 1, 2 or even 3, of cache), or need more memory pages than the TLB can address, the whole matrix would have to be reloaded at each BLAS call. And in the original scheme with *recopies* or with *slice* storage the number of columns increases strongly with the height of the octree. That's why the multipole and local matrices also have to be treated by blocks: we split them in sub-matrices with a constant number of columns, namely $NbExp_{max}$, and the same number of rows than the original matrix. As described in figure 4.7, the i^{th} sub-matrix of the local expansion matrix is computed by a matrix-matrix product between the $M2L$ transfer matrix and the i^{th} sub-matrix of the multipole expansion matrix. A matrix-matrix product with $NbExp$ expansions is then computed as $NbExp_{max}/NbExp$ matrix-matrix products with $NbExp_{max}$ expansions, plus the last columns corresponding to the remainder of the division. This decomposition of the matrix-matrix product is hereafter named "$NbExp_{max}$ decomposition".

Optimal values for $NbExp_{max}$ will be experimentally determined in section 4.4.1.

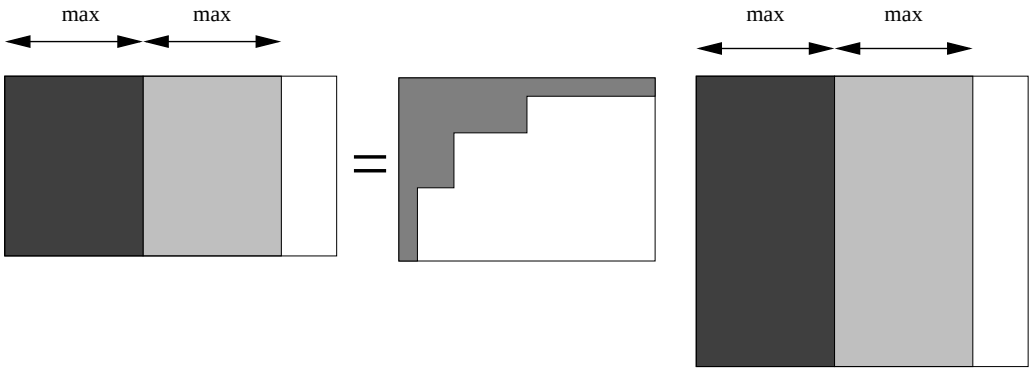


Figure 4.7: $NbExp_{max}$ decomposition (max denotes $NbExp_{max}$). This matrix-matrix product is computed as several matrix-matrix products with at most $NbExp_{max}$ columns in the multipole and local matrices.

Remark 4.4. *One can argue that the choice to browse once the local expansions and several times the multipole expansions as explained page 39, combined with a column storage of our matrices, leads here to a leading-dimension problem for the multipole and local matrices: we access these matrices mainly by rows while they are stored by columns. It is certainly possible when considering the transposed matrix-matrix product with a leading dimension equal to $NbExp_{max}$ for the multipole and local matrices to solve this problem but this was not used since it prevents the easy concatenation of the expansions as columns of one matrix. Moreover this leading dimension problem might not heavily affect the BLAS efficiency.*

Remark 4.5 (BLAS for $M2M$ and $L2L$ operators). *As exposed in [HJ96], $M2M$ and $L2L$ can easily be computed with level 3 BLAS calls: when $M2M$ and $L2L$ are written as matrix-vector products between a cell expansion and the expansion of its father, all cells of the same type of child share the same $M2M$ and $L2L$ transfer functions. Several expansions can thus be computed all at once with level 3 BLAS calls when grouping them either with recopies or with an appropriate memory storage that avoids recopies (row and other data storages used for $M2L$ cannot be used here because the storage cannot be the same at the father level and at the child level).*

4.4 Tests and comparisons

Performance tests have been performed in order to validate the BLAS implementation, along with its parameters, that leads to the fastest $M2L$ computation. These tests have been performed on one processor: either one IBM Power3-II WH2+ (375 MHz, 1.5 GFLOPS, L1 cache size: 64 KB, L2 cache size: 4 MB, and 2 GB of memory), or one IBM Power4+ (1454 MHz, $\approx$ 6 GFLOPS, L1 cache size: 32x2 KB, L2 cache size: 1.41 MB, L3 cache size: 32 MB, and 8 GB of memory, thus requiring 64-bits compiler mode). These are located at LaBRI[4]. We have also used Power3 NH2 processors (L1 cache size: 128 KB, L2 cache size: 8 MB, and 16 GB of memory) at CINES[5].

The number of particles used for the simulation is not usually precised since the runtimes here measured depend only of the height of the unifom octree and of P, but of course the height used implies a range in the number of particles that balances the near field and the far field computations.

First of all we emphasize on the use of level 3 BLAS instead of level 2 ones with figure 4.8: for an octree of size 5 and double height $M2L$ kernel the use of *recopies* clearly improves the performances. This is also valid for single height $M2L$ kernel and for other heights. For the lowest values of P, no substantial gain is however obtained but this will be done with *row* and *slice* storages as explained in section 4.4.2.

We will now determine the best decomposition for the sparse $M2L$ transfer matrix in the single height kernel case, along with the optimal $NbExp_{max}$ values, and then compare the different schemes that enable level 3 BLAS usage.

4.4.1 Decomposition for single height $M2L$ kernel

In this section, we will first see how we have established, for each architecture (Power3 or Power4), the best $NbExp_{max}$ value (see section 4.3.3) for the *band_ blas*, *cband_ blas* and *block_ blas* routines that implement the

[4]Laboratoire Bordelais de Recherche en Informatique, Talence, FRANCE.
[5]Centre Informatique National de l'Enseignement Supérieur, Montpellier, FRANCE.

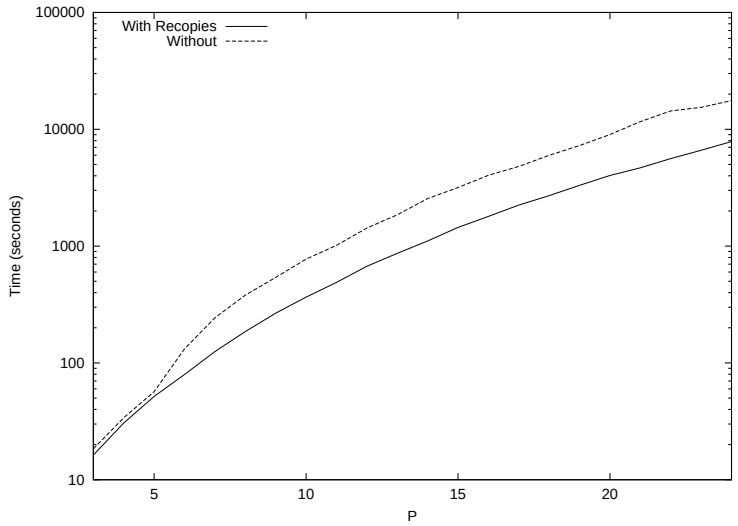


Figure 4.8: Downward pass CPU times with an octree of height 5, *full_ blas* method (double height kernel), with and without *recopies*.

	$level = 4$	$level = 5$	$level = 6$	$level = 7$
recopies	216	2744	27000	238328
row storage	6	14	30	62
slice storage	46	222	958	3966

Table 1: $NbExp$ values according to our different schemes (with FBC, and $ws = 1$, see section 4.3.2).

corresponding decompositions. For *cband_ blas* routine the best dim_{cband} value had also to be established, while for *block_ blas* routine we had also to determine the best threshold value, denoted by s_{block}, that determines the terminal case of the recursion (see section 4.2.2). With these parameters, we will then be able to compare the *band_ blas*, *cband_ blas* and *block_ blas* routines, and finally emphasize on the interest of the $NbExp_{max}$ decomposition. All these tests have been performed on one single matrix-matrix product, corresponding to $M2L$ operation, for different values of P.

First we need to determine the optimal $NbExp_{max}$ values. These are clearly machine dependent, and they depend on P too since P determines the number of rows in the multipole matrix and in the local matrix. Depending on the level 3 BLAS scheme used and on the level in the octree, the number of multipole and local expansions concatenated, denoted by $NbExp$, differs significantly. For performance tests with different $NbExp_{max}$ values, we have only considered $NbExp$ values for cells with complete interaction list: matrix-matrix products for cells with incomplete interaction list have very varying values for $NbExp$ but they are outnumbered compared to matrix-matrix products for cells with complete interaction list. The table 1 shows $NbExp$ values according to our different schemes for cells with complete interaction list.

In practice tests for optimal $NbExp_{max}$ values will be performed, according to P, for one single "big enough" $NbExp$ value, denoted $NbExp_{ref}$. Indeed, as confirmed by more complete tests, these optimal values for $NbExp_{ref}$ will also be among the optimal values for higher $NbExp$ values (corresponding to higher heights), while the lower $NbExp$ values will not require any $NbExp_{max}$ decomposition (unless P is great, and then best $NbExp_{max}$ values found for $NbExp_{ref}$ are also optimal). For Power3 architecture, search can thus be performed for $NbExp_{ref} = 958$ ($level = 6$ and *slice* storage) while for Power4 optimal values for $NbExp_{max}$ higher than 958 can be found thus requiring a greater $NbExp_{ref}$, namely 2744 ($level = 5$ and scheme with *recopies*). In practice when P ranges from 3 to 30, optimal values found for $NbExp_{max}$ usually range from several hundreds for low values of P, up to a few dozens for high values of P: see for example figure 4.9.

For *cband_ blas* and *block_ blas* routines, this search is coupled with a search of respectively the best dim_{cband} and the best s_{block}: all pairs are tested to find the best one. However results have shown that $NbExp_{max}$ is mainly independent from dim_{cband} or s_{block}: when splitting both multipole and local matrices according to $NbExp_{max}$, the sub-matrices size depends indeed only on P (for the number of rows) and $NbExp_{max}$ (for the number of columns): see figure 4.7.

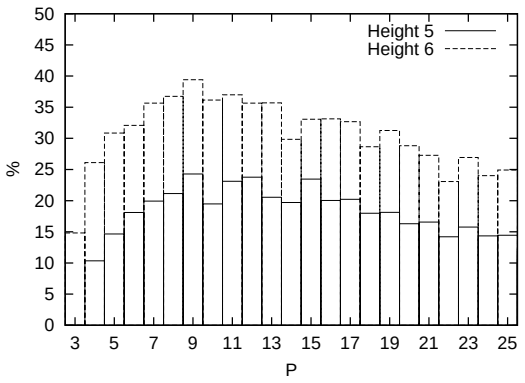
For dim_{cband}, tests have also shown that one (or two close ones) values are optimal, and these are independent of P. Once the optimal height of a *band* is determined, it corresponds indeed to the optimal size of the square blocks used by $ZGEMM$ routine and this remains unchanged when the $M2L$ transfer matrix get bigger.

For the first values of P (roughly $P = 7$ in our tests) best s_{block} values equal P which means it is better not to use the recursive decomposition: we are therefore brought back to the *band_ blas* decomposition. However when P is higher, it is faster to use the recursive decomposition and fastest computations are then found for low values of s_{block} (between 2 and 5 in our tests): see figure 4.9.

Now that the optimal values of these parameters have been found for each value of P, we can compare *band_ blas*, *cband_ blas* and *block_ blas* routines. It appears that *cband_ blas* is faster than *band_ blas* only for high values of P ($P \geq 14$ on the IBM Power3): the zeros added in the *cband_ blas* decomposition are too much costly unless the $M2L$ transfer matrix is very big. For *block_ blas* routine the optimal values found for s_{block} make its computation as fast as *band_ blas* routine for low values of P and faster for higher values, as explained above. And for these higher values of P, *block_ blas* routine is faster than *cband_ blas* routine.

In conlusion we thus choose the *block_ blas* routine for BLAS computation of $M2L$ operator with single height kernel. However the gains and losses here mentioned are all lower than 5 % of the *band_ blas* routine runtime: this does not really justify the implementation of the more complex *cband_ blas* and *block_ blas* routines.

Before comparing the different schemes used with level 3 BLAS in the next section, the relevance of the $NbExp_{max}$ decomposition is shown on figure 4.9 for the scheme with *recopies*: for each value of P, we show the gain for the downward pass CPU times due to the $NbExp_{max}$ decomposition. With growing octree height, the numbers of expansions $NbExp$ treated with one single matrix-matrix product increases, and the gain offered by the $NbExp_{max}$ decomposition increases thus too. With *row* and *slice* storages, $NbExp_{max}$ decomposition will be likewise relevant when the $NbExp$ value exceeds the optimal $NbExp_{max}$ value.



Gain in percentage.

P	s_{block}	$NbExp_{max}$	P	s_{block}	$NbExp_{max}$
3	3	921	15	5	102
4	4	936	16	3	96
5	5	463	17	3	84
6	5	403	18	3	84
7	5	307	19	4	102
8	3	260	20	7	126
9	7	210	21	3	120
10	2	168	22	2	120
11	10	156	23	5	132
12	10	126	24	6	120
13	1	138	25	11	120
14	1	120			

s_{block} and $NbExp_{max}$ values (Power4).

Figure 4.9: Gain in percentage for the full downward pass CPU times offered by a computation with $NbExp_{max}$ decomposition relative to one without $NbExp_{max}$ decomposition. Tests performed on IBM Power4, using the *block_ blas* routine (with optimal s_{block}) and the scheme with *recopies* for an octree height equal to 5 ($NbExp = 2744$) and 6 ($NbExp = 27000$). The corresponding values for s_{block} and $NbExp_{max}$ are also given.

4.4.2 Recopies and special data storages

In order to compare the scheme with *recopies* with *row* and *slice* storages, CPU times have been measured on the full downward pass of the FMM. Two combine effects appear:

- As mentioned in [HJ96], the cost of copying vectors relatively to the cost of the matrix-matrix product decreases for growing values of P: copies of multipoles and local expansions are indeed performed in $\mathcal{O}(NbExp \times P^2)$ while the matrix-matrix product requires $\mathcal{O}(NbExp \times P^4)$ operations. This is more obvious with double height kernel than with single height kernel since the amount of computation for the matrix-matrix product is much more costly with double height, while the cost of *recopies* is the same. Figures 4.10 and 4.11 show the gain of *row* and *slice* storages relative to the scheme with *recopies* according to different values of P for an octree height of 6: *row* and *slice* storages offer thus better gains relative to *recopies* for low values of P, and these gains are always higher for single height kernel.

- These gains are also influenced by the height of the octree: with growing heights of the octree, the number of expansions treated with one matrix-matrix product increases for *row* and *slice* storages (see table 1) and the proportion of *blank boxes* decreases for *slice*. In figure 4.11, we have only $NbExp = 30$ for *row* storage which leads for double height kernel and $P \geq 11$ to worse performance than *recopies* and its $NbExp = 27000$. For low heights such as 4, the use of *recopies* is even a little bit faster for all values of P since it is the only one to allow enough $NbExp$ to be treated all at once.

Slice storage is here more efficient than *row* storage but this is mainly due to the too small $NbExp$ for *row* storage at height 6 (i.e. $NbExp = 30$) while for *slice*, $NbExp = 958$ allows better BLAS 3 efficiency: indeed tests performed for $H7$ on IBM Power3 with 16GB memory (CINES) shows that *row* storage is a little bit faster for this height. In fact *slice* storage does not usually offer gain relative to *row* storage and moreover *slice* storage needs very long consecutive memory areas that cannot be allocated for too high values of P or too high heights when memory allocation with *row* storage may succeed. For these reasons we favor *row* storage, and since the $NbExp$ values enabled by *slice* storage seem to be high enough, *level* storage has not been implemented.

In conclusion *row* and *slice* storages offer both obvious gains relative to the scheme with *recopies* for low and mean values of P and consequent heights. Moreover, we will generally prefer *row* data storage. We however keep the implementation with *recopies* since it may be more suitable for the parallelisation on distributed memory architectures as well as for the adaptive version of the FMM (in this last case it may however be rewritten in the same way as for cells with incomplete interaction list, see section 4.3.1).

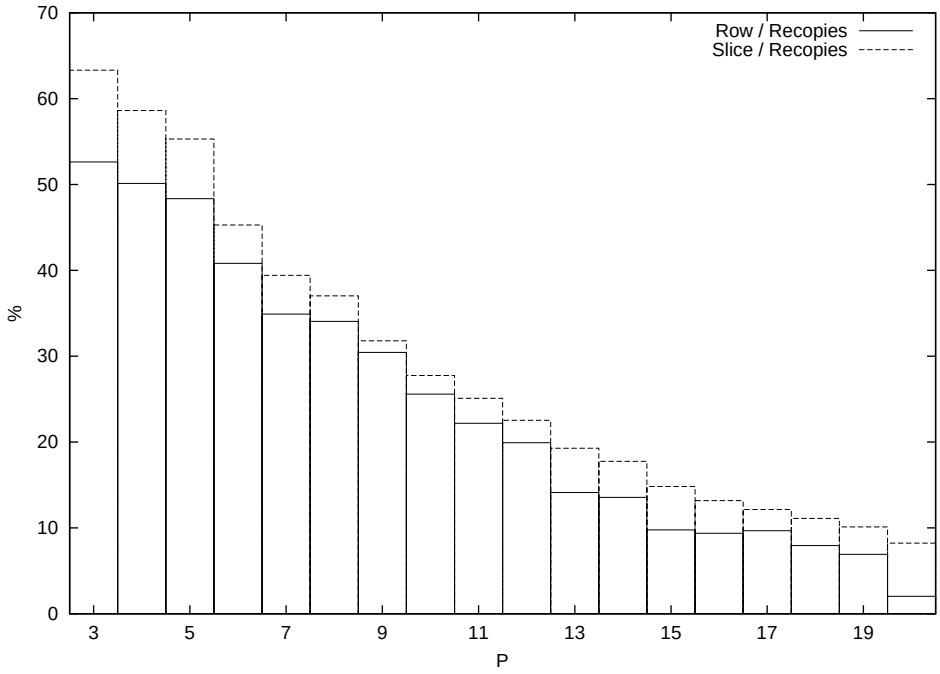


Figure 4.10: Gain in percentage offered by *row* and *slice* storages relative to the scheme with *recopies* for downard pass CPU times with an octree of height 6 and *block_ blas* routine (single height kernel). Tests performed on IBM Power4.

In summary, the best BLAS implementations are therefore *block_ blas* and *full_ blas* (depending on the $M2L$ kernel height), with *row* data storage.

4.5 Memory requirements

We here present the memory requirements for the BLAS computation of $M2L$. First of all, multipole expansions need to be converted in multipole expansion vectors whose size is: $\sum_{j=0}^{P} \sum_{k=-j}^{j} c = (P+1)^2 c$, where c denotes the size of a complex number (see section 1.2.3). Then the $M2L$ transfer functions are converted in $M2L$

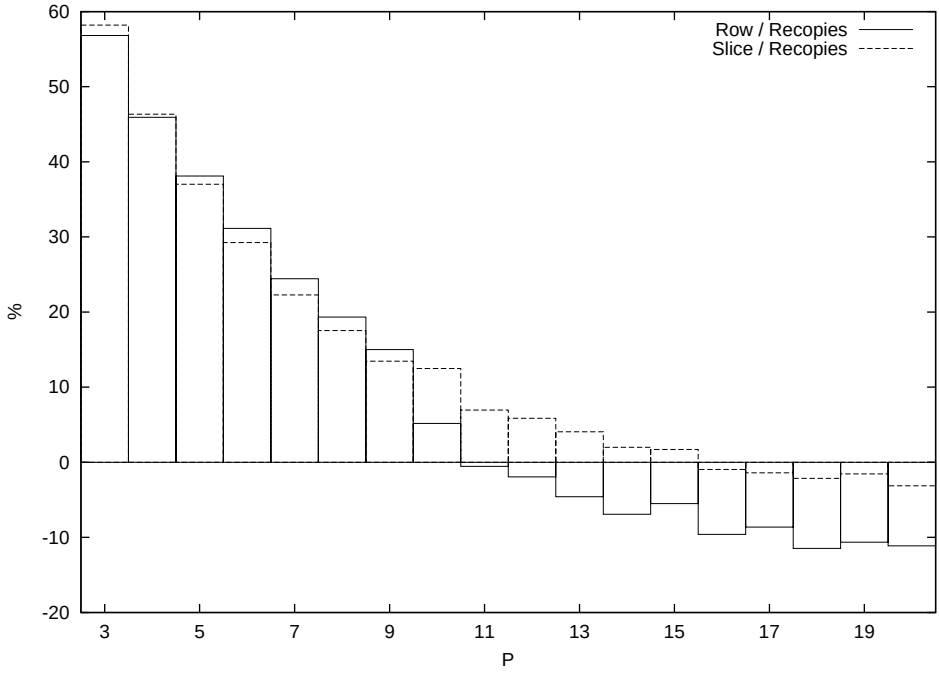


Figure 4.11: Gain in percentage offered by *row* and *slice* storages relative to the scheme with *recopies* for downard pass CPU times with an octree of height 6 and *full_blas* routine (double height kernel). Tests performed on IBM Power4.

transfer matrices. For double height $M2L$ kernel, the size of these matrices is:

$$\mathcal{T}_{matrix} = \frac{(P+1)(P+2)}{2}(P+1)^2 c.$$

This can become quite big for high values of P since it grows as $\mathcal{O}(P^4)$, against $\mathcal{O}(P^2)$ for the classic $M2L$ computation. For single height kernel, we consider here matrices decomposed according to *block_blas* decomposition. Their memory usage is however the same as with *band_blas* decomposition which can be computed as follows: the *bands* being numbered from the top to the bottom of the matrix, the *band* with number M, $M \in [\![0, P]\!]$, has $M+1$ rows and $\sum_{N=0}^{P-M}(2N+1) = (P-M+1)^2$ columns, and the memory usage of all *bands* is therefore:

$$\mathcal{T}_{matrix} = \sum_{M=0}^{P}(M+1)(P-M+1)^2 c = \frac{(P+1)(P+2)^2(P+3)}{12}c.$$

The memory usage for *cband_blas* decomposition is a little bit higher due to the presence of some zeros in the *bands*.

With level 2 BLAS no other memory is needed. But with level 3 BLAS, we need extra memory. First of all for the cells with incomplete interaction list, we need two temporary buffers: one for the multipole matrix and the other for the local matrix. The number of expansions recopied in such buffer varies greatly and can become great with growing heights of the octree: for example with a height of 6, it ranges from 1 to 37952. However the computation can be as efficiently performed with smaller buffers than the maximum needed: if the size of the two buffers enables $\mathcal{N}_{Buf}$ expansions to be treated alltogether in one single matrix-matrix product and if n pairs of multipole / local expansions share the same $M2L$ vector, with $n > \mathcal{N}_{Buf}$, they are treated with $\frac{n}{\mathcal{N}_{Buf}}$ consecutive calls, plus one call for the remaining expansions. Provided than $\mathcal{N}_{Buf}$ is high enough to reach optimal level 3 BLAS efficiency, the performances will be the same as with buffers enabling n expansions. In practice, $\mathcal{N}_{Buf}$ values are the same as the $NbExp$ values discussed in section 4.4.2 that gives the best efficiency: concretely a few thousands of expansions are enough for all values of P.

For cells with complete interaction list, we have to distinguish the scheme with *recopies*, *row* storage and *slice* storage. With *recopies* the memory requirements are the same as for cells with incomplete interaction list: we need two buffers for the *recopies* of the multipole and the local matrices. For optimal level 3 BLAS efficiency, we only impose that they can at least contain a few thousands of expansions. We consider that these buffers

can be the same as the ones used for the cells with incomplete interaction list and thus that the scheme with *recopies* does not need supplementary memory.

The *row* storage does not require any temporary buffer but with growing octree heights, it necessitates longer consecutive memory areas for each row. This is even more critical with *slice* storage since it needs even longer consecutive memory areas for each slice. Moreover the *blank boxes* introduced with *slice* storage require extra memory. Concretly each blank expansion has the same size as a local expansion and their number can be computed as follows (see also section 4.3.2): at level l, for cells with complete interaction list and for each of the eight *types of child*, there is $2^{l-1}-2.ws$ slices with $2^{l-1}-2.ws$ rows in each slice. We have then 1 *blank box* at the end and at the beginning of each row, minus the first and the last *blank box* of the slice, which gives $2*(2^{l-1}-2.ws)-2$ blank expansions per slice. The number of all blank expansions $\mathcal{N}_{\mathcal{B}}$ for an octree of height H is therefore $\sum_{l=3}^{H} 8(2*(2^{l-1}-2.ws)-2)(2^{l-1}-2.ws)$ (there is no cells with complete interaction list for levels $l \leq 2$), which results for $ws=1$ in:

$$\mathcal{N}_{\mathcal{B}} = \sum_{l=3}^{H} 16(2^{l-1}-3)(2^{l-1}-2) = \frac{16}{3}.4^H - 80.2^H + 96.H + \frac{128}{3}.$$

To summarize, the scheme with *recopies* and *row* storage have the same memory requirement, namely:

$$\begin{aligned} Mem_{Sg}(P,H) = \quad & \mathcal{N}(H)\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)c + \mathcal{N}_{\mathcal{T}}\mathcal{T}_{matrix} \\ & + \mathcal{N}_{Buf}\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)c, \end{aligned} \tag{4.5}$$

and with *slice* storage we have:

$$\begin{aligned} Mem_{Db}(P,H) = \quad & \mathcal{N}(H)\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)c + \mathcal{N}_{\mathcal{T}}\mathcal{T}_{matrix} \\ & + \mathcal{N}_{\mathcal{B}}\tfrac{(P+1)(P+2)}{2}c + \mathcal{N}_{Buf}\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)c. \end{aligned} \tag{4.6}$$

4.6 Study of recopies and special data storages for other $M2L$ improvements.

One could argue that some speedup might be as well obtained for FFT and rotations when using *recopies* or data storages such as those described above.

We have thus implemented the scheme with *recopies* and *row* storage for FFT. Tests were performed for a block version of the FFT with the usual block size 4. First it appears that the use of *recopies* dramatically degrades the performances: indeed the cost of *recopies* for block FFT is at least twice the one for BLAS computation since the size of the arrays for the FFT is $2(P+1)^2$ (single height kernel), compared to $(P+1)^2$ for multipole vectors and $\frac{(P+1)(P+2)}{2}$ for local expansion vectors. Moreover this cost is not amortized by the large-grain correlation in $\mathcal{O}(P^3)$ as it was with the BLAS computation due to the matrix-matrix product in $\mathcal{O}(P^4)$. At last besides longer pipelines, the only gain that might be expected when grouping several $M2L$ pairs is to avoid the reload of the array containing the $M2L$ transfer function in Fourier space between two $M2L$ computations: potentially, there is here no great gain. It has also to be noted that such schemes imply one array for the local expansions in Fourier space for every cell of the level which severely increases the memory needs.

Nevertheless, we have also tested FFT with *row* storage which avoids the costful *recopies*. The cells with incomplete interaction list which require more *recopies* for their local expansions were computed as usual. Here no slowdown was measured, but no gain either.

As explained in section 3.3, we cannot use BLAS to speed up the $M2L$ computation with rotations. Without BLAS calls it is then likely that no speedup will be obtained for rotations when grouping several $M2L$ pairs with either *recopies* or *row* storage.

5 Comparison of the $M2L$ improvements

5.1 Memory requirements

We first compare the memory requirements of the different schemes since it may be the first choice criterion. We here focus only on the memory used for the expansions: the memory used for the particles remains indeed unchanged when computing $M2L$ operation differently. Using theoretical estimations established in (1.14), and (1.15) for the classic $M2L$ computation, (2.9) and (2.11) for the $M2L$ computation with FFT, (3.19) and (3.20) for the one with rotations, and (4.5) and (4.6) for the BLAS computation, we have plotted in figure 5.1, for each scheme, the ratio of its memory need to the classic $M2L$ memory need, using an octree of height 6 and only multiples of the FFT block size. For BLAS computations, we have used $\mathcal{N}_{Buf} = 5000$ (see section 4.5) which is surely big enough for optimal BLAS efficiency.

While the rotation requirements are almost unnoticeable, and the BLAS ones remain moderate, FFT extra memory appear as problematic, especially for double height $M2L$ kernel. The same ratios apply for other FFT block sizes. Moreover, the ratio of 2 for BLAS in the double height kernel with high values of P is due to the dense $M2L$ transfer matrices (see section 4.5): for higher octree heights this ratio remains close to 1.5, since the number of $M2L$ functions is constant while the number of cells in the octree grows exponentially. At last, the additional cost of the *blank boxes* in *Slice_rearrange* over *Row_rearrange* is very small.

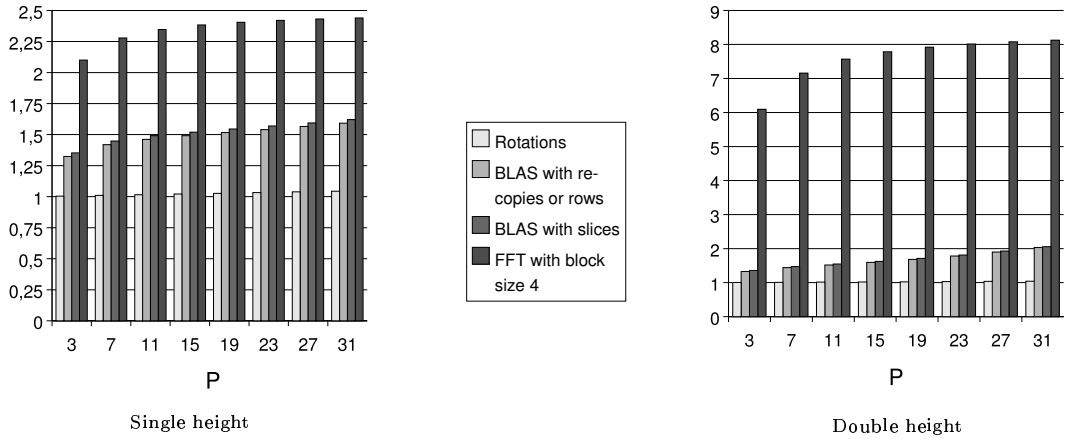


Figure 5.1: Memory requirements of the different $M2L$ computation schemes for an height of 6 (ratios to classic $M2L$ computation requirements).

5.2 CPU times for single height $M2L$ kernel

Figure 5.2 compares the different schemes for an octree of size 5. If the BLAS version outperforms the classic $M2L$ version and the one with rotations, the block FFT is faster, especially for values of P higher than 10. However we recall that, without rescaling of the particle coordinates, in this test the FFT with block size 4 is unstable for $P \geq 14$, and the one with block size 3 for $P \geq 23$ (see table 2.10).

When focusing on low and medium precisions, we recall also that for growing heights of the octree the BLAS computation with *Row_rearrange* or *Slice_rearrange* becomes more efficient since the length of rows and slices increases at the leaf level which represents most of the runtime: the number of expansions treated in one level 3 BLAS call thus increases, as well as the efficiency of the BLAS. Figure 5.3 shows that BLAS is clearly competitive with the FFT with block size 4, in an octree of height 7 and for low and medium precisions. The memory requirements of the FFT improvement are problematic here, since on figure 5.3, the brutal increase in runtime for FFT at $P = 10$ is due to swapping despite the 16 GB of memory available.

5.3 CPU times for double height $M2L$ kernel

With double height kernel, the use of BLAS clearly outperforms all other methods. Figure 5.4 compares the different schemes for an octree of height 5. For the BLAS we have plotted only the *row* data storage, but *slice* storage and the scheme with *recopies* have similar performances. For the FFT, the block version with blocks of size 4 have been plotted; with our tests, it is only stable for $P \leq 8$ (see section 2.8), and more stable FFT

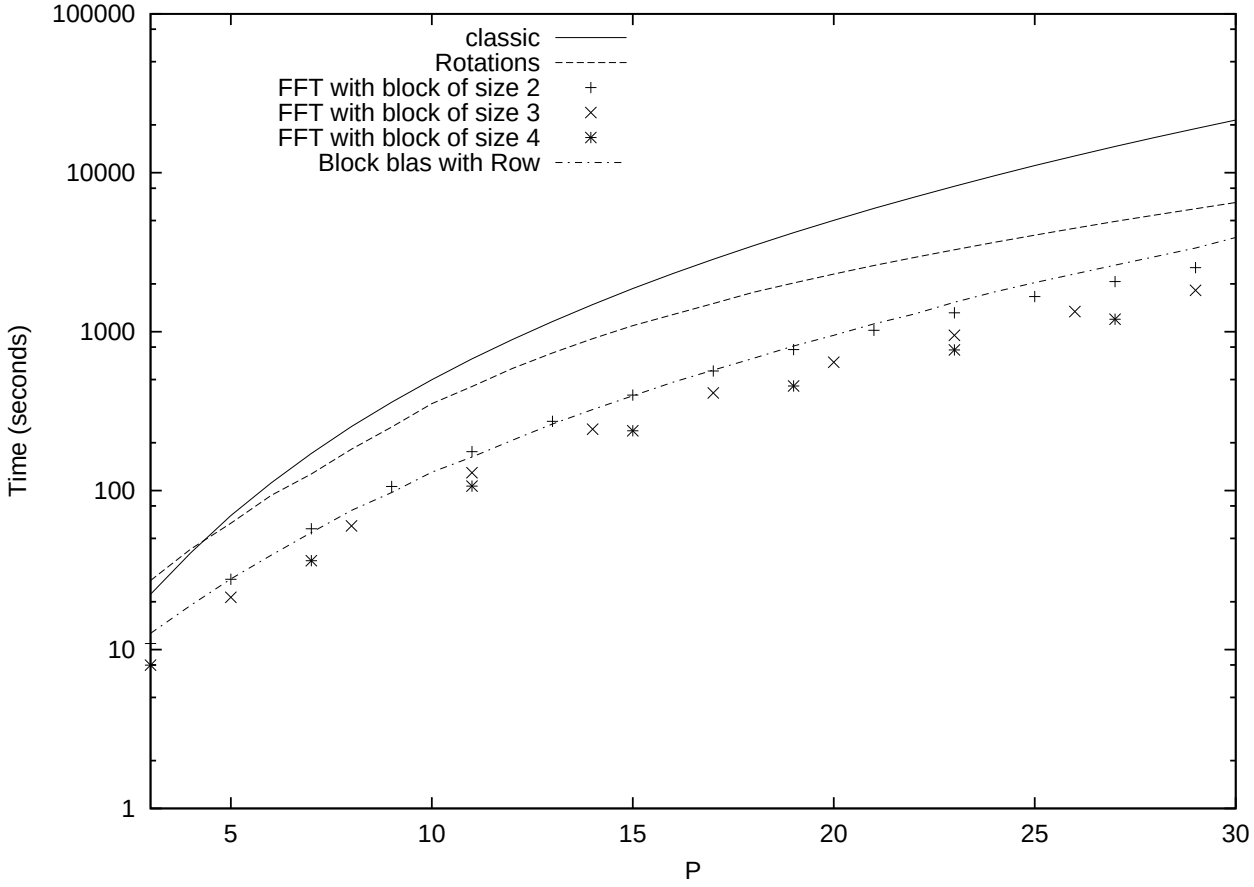


Figure 5.2: Logarithmic downward pass CPU times of the different $M2L$ computation schemes, for an octree of height 5 and with single height $M2L$ kernel. Tests are performed on an IBM Power3 with 2 GB of memory.

with lower block sizes are slower. Moreover, we have stop the tests at $P = 19$ since on the IBM Power3 (where the tests were performed), the 2GB of memory were insufficient for all FFT block sizes with $P > 19$. However it must be noticed that at this octree height, the rotation scheme becomes faster than all the BLAS schemes for $P \geq 23$: this is of course the aftermath of the lower operation count for the rotation scheme. Finally, as for single height kernel, the BLAS computation with *Row_ rearrange* and *Slice_ rearrange* become even more efficient for higher heights of the octree.

5.4 Computational efficiency

In order to illustrate the efficiency of the BLAS versions that makes them faster than the other schemes, we present the table 2 that shows the MFLOPS (Millions of Floating Point Operations per Seconds) when computing $M2L$. The results are shown as percentages of the peak performance of the IBM Power3 used: 1500 MFLOPS. They have been computed with the Hardware Performance Monitor (HPM) Toolkit (version 2.4.3) as the average MFLOPS rate over all $M2L$ computations of a full FMM computation. For level 3 BLAS, we have used *block_ blas* and *full_ blas* routines for respectively single and double height kernels, with an octree of height 5 and with *recopies* (the additional cost of copying the expansions is not considered here since we focus on the BLAS routine efficiency). It takes into account all cells, i.e. with and without complete interaction list. The height of the octree does not matter for classic $M2L$, rotations and FFT. These results can however not be directly compared from one row to the other: we recall here that the operations count differs! However it clearly illustrates how efficiently the processor is used in the different implementations and why BLAS computations outperform the other schemes.

This table also indicates that our decomposition of the matrix-matrix product for single height kernel in the *block_ blas* routine, though satisfactory, is not optimal when compared to the *full_ blas* routine efficiency of the double height kernel, especially for low values of P.

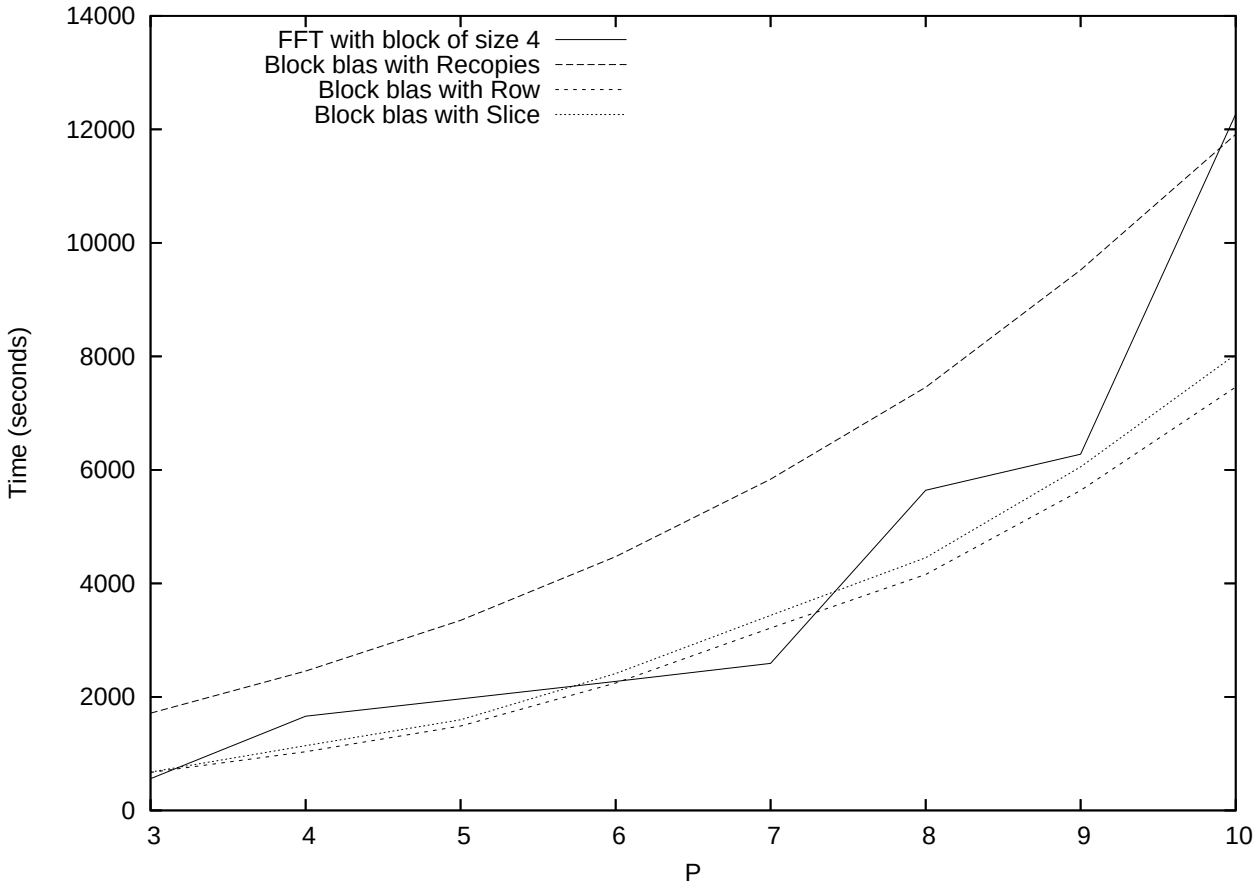


Figure 5.3: Downward pass CPU times of the different $M2L$ computation schemes, for an octree of height 7 and with single height $M2L$ kernel. Tests are performed on an IBM Power3 with 16 GB of memory (CINES).

	Single height $M2L$ kernel		Double height $M2L$ kernel	
	$P = 7$	$P = 15$	$P = 7$	$P = 15$
classic	9.4 %	16.3 %	14.5 %	18.3 %
rotations	5.4 %	7.8 %	8.4 %	10.9 %
FFT with block size 4	4.5 %	13 %	12.4 %	18.4 %
level 3 BLAS	46.4 %	74 %	85.9 %	89.2 %

Table 2: Computational efficiencies of the different $M2L$ computation schemes.

5.5 Single versus double height kernels

We have already seen (see theorem 4) that a sharp error bound has been theoretically proven only for the single $M2L$ height kernel. The double height $M2L$ kernel is certainly more precise, and for a given precision it would therefore requires a lower value for P than the single height one. But since no precise error bound is available for double height kernel, we cannot a priori know what this lower P will be.

We thus compare here CPU times with practical accuracies for both kernel heights. As already exposed (see for example [WHG94] for double height $M2L$ kernel, and [Ran99] for single height kernel), these practical accuracies are better than the theoretical ones which correspond to worst-case errors. These worst-case errors are indeed obtained with spherical regions, while we use in fact smaller cubical cells because of the octree. Moreover, we use relative error instead of absolute one so that the results are problem independent, and these relative errors are computed for the potential and not for the force components because these latters can be almost null. Since some discontinuities appear in the potential error for particles crossing cell boundaries (see

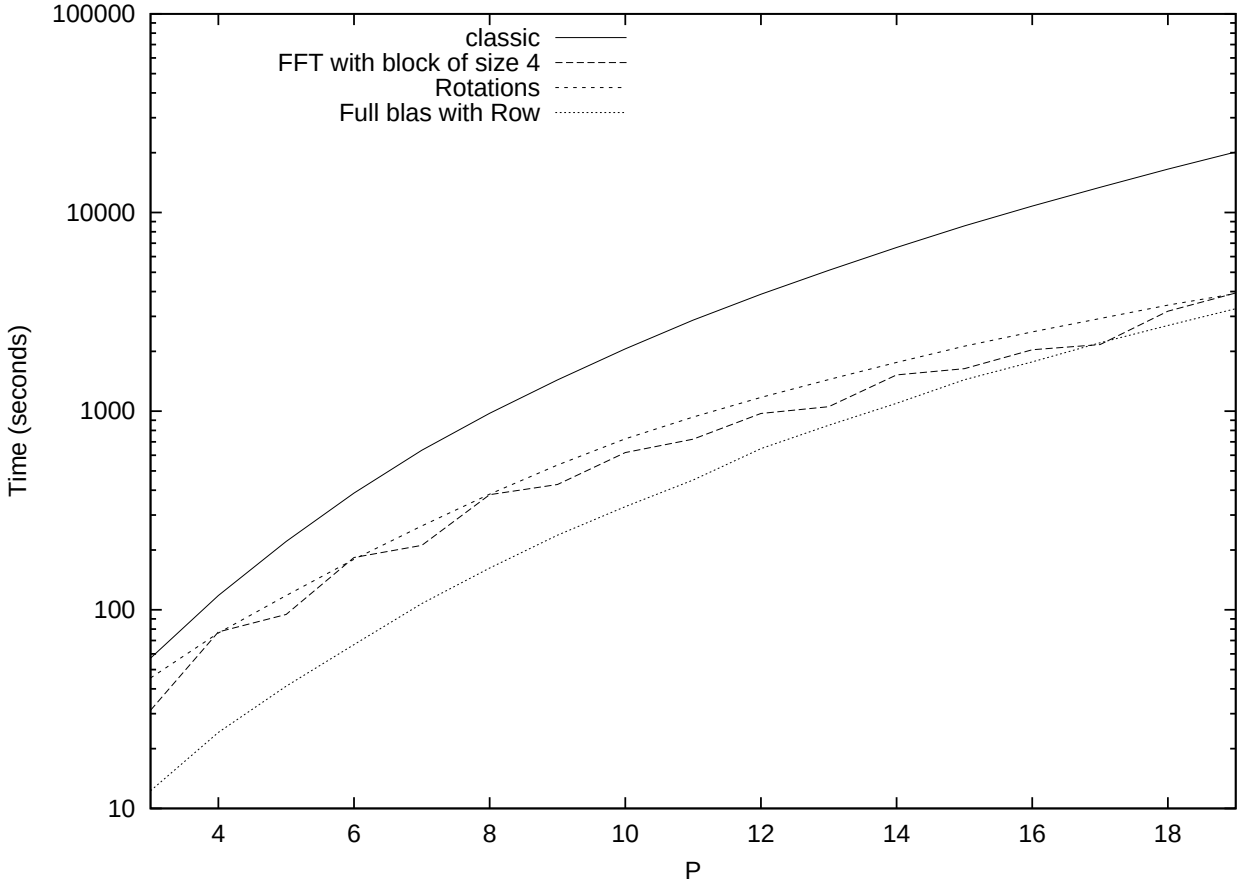


Figure 5.4: Logarithmic downward pass CPU times of the different $M2L$ computation schemes, for an octree of height 5 and with double height $M2L$ kernel. Tests are performed on an IBM Power3 with 2 GB of memory.

[Ran99]), we choose to study the more stable RMS average error instead of the maximum error. This RMS average error ε_{rms} is computed as follows:

$$\varepsilon_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\Phi_{Dir}-\Phi_{FMM}}{\Phi_{Dir}}\right)^2}.$$

We consider here a gravitational potential computed for an uniform distribution with 100000 bodies and an octree of height 4, which results in 25 bodies per leaf in the mean. We recall that when the number of bodies per leaf increases, the part of the near field (directly computed) in the potential becomes higher and the potential becomes thus more precise. Figure 5.5 presents, for each $M2L$ scheme, the tradeoff between practical error and CPU times (downward pass only) with both single and double height kernels. The scales are logarithmic, and the values of P plotted for double height kernel range from 3 to 14 with a step of 1, while for single height kernel they range from 3 to 29 with step 2: $P = 14$ in double height and $P = 29$ in single height correspond indeed to the same accuracy (precisely the first accuracy below 1.0×10^{-9} in our simulation).

As far as classic $M2L$ is concerned, the single height kernel is more efficient for low precisions and the double height one for high precisions. This slightly differs from the results of [Ell95] (section C.3.1) where a brief comparison using theoretical operation count vs. accuracy shows that both heights are theoretically as efficient.

And as theoretically justified in sections 1.2.4, 2, 3 and 5.4, while rotation and BLAS computations favor the double height kernel, the FFT improvement is generally more efficient with single height kernel.

Therefore we still have to compare the best of each scheme: this is done in figure 5.6 where we present FFT and BLAS versions on the same plot (classic and rotations have already been discarded in sections 5.2 and 5.3). The BLAS in double height kernel are then clearly more efficient than the FFT with single height kernel: by

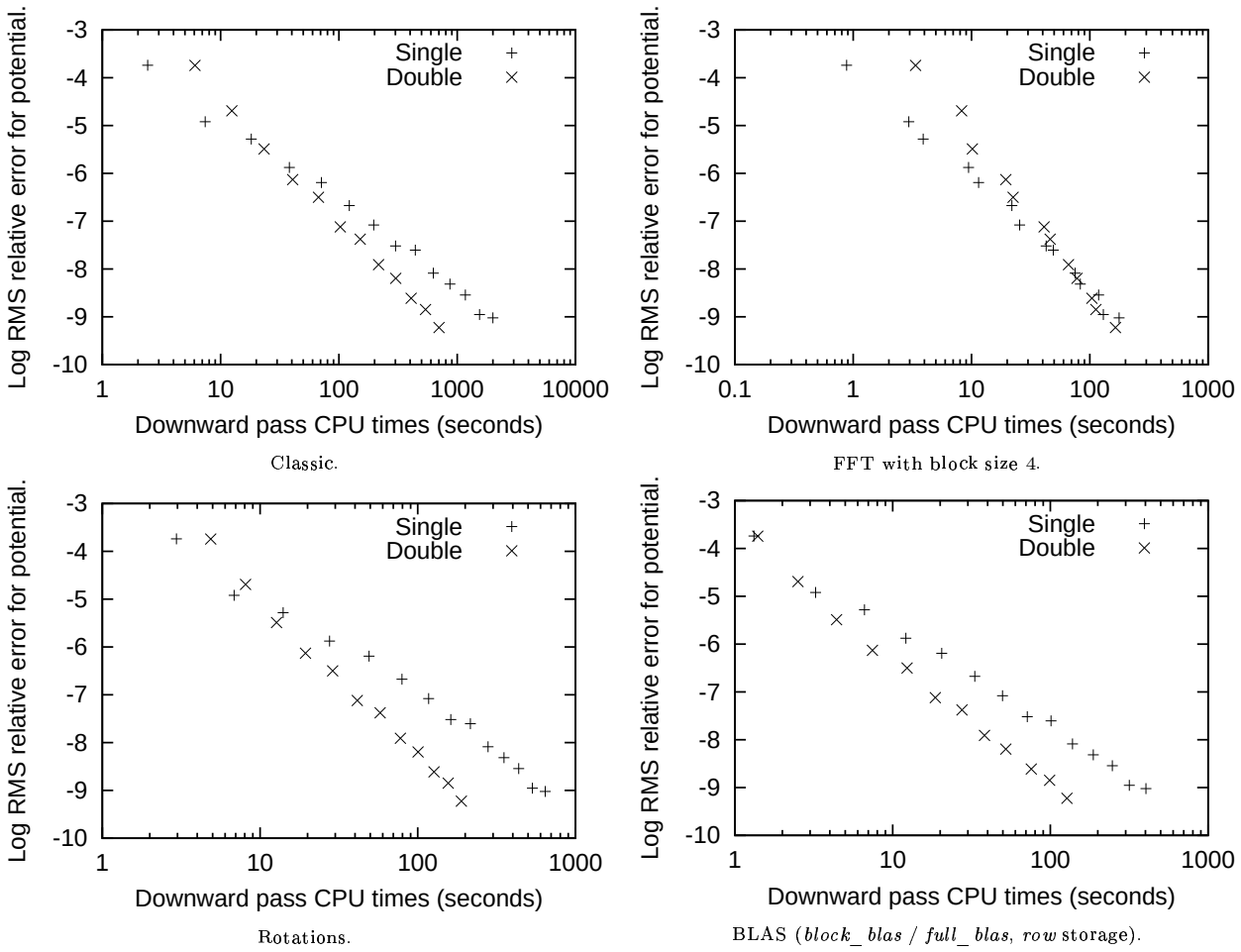


Figure 5.5: Tradeoffs between practical accuracies and CPU times for single and double height kernels.

examples, for a practical error below 1.0×10^{-6} (respectively 1.0×10^{-8}) the gain is 35% (respectively 30%). This fully validates the relevance of our new BLAS version compared to the previous $M2L$ improvements.

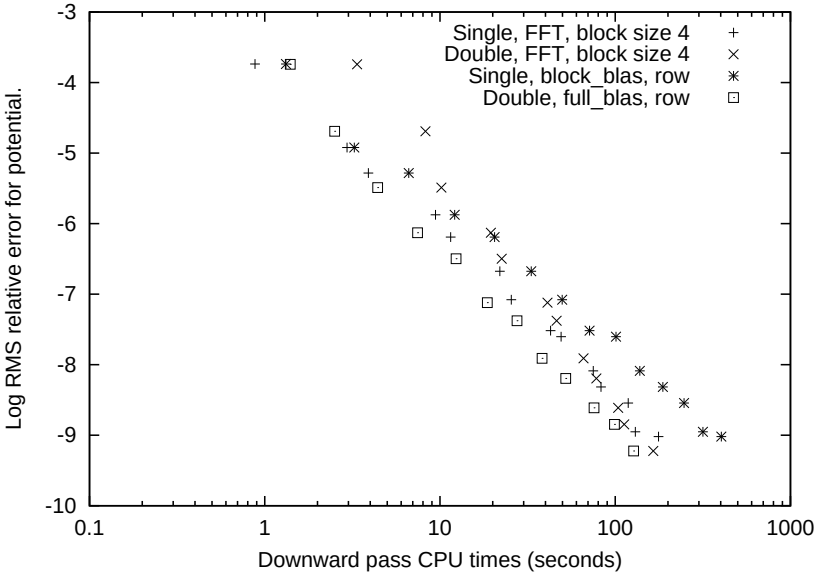


Figure 5.6: FFT block size 4 vs. BLAS (*block_ blas* / *full_ blas*, *row* storage).

6 Conclusion

In this paper, we have presented an overall study of the best implementation of the Fast Multipole Method for the serial computation of gravitational or electrostatic simulations of uniform distributions. A detailed study of the error bound has lead us to two expressions of the $M2L$ operator that converts a multipole expansion into a local expansion: while *double height $M2L$ kernel* is generally more efficient, only *single height $M2L$ kernel* ensures a sharp error bound.

For each $M2L$ expression, we have efficiently implemented different schemes that speed up $M2L$ computation: schemes such as *FFT with blocks* and *rotations* allow a reduction of the theoretical operation count from $\mathcal{O}(P^4)$ to $\mathcal{O}(P^3)$, while the introduction of BLAS (Basic Linear Algebra Subprograms) greatly quickens the $\mathcal{O}(P^4)$ computation.

To our knowledge, this is the first implementation of the FFT enhancement for double height $M2L$ kernel. We have also gone into details of the implementation and we have shown that numerical instabilities remain even in the block version of this improvement. The rotation scheme has been presented in details according to the formulae used in our FMM implementation.

As an alternative, a BLAS (Basic Linear Algebra Subprograms) version has been proposed for the dense matrices of the double height kernel as well as for the sparse matrices of the single height one. A scheme with *recopies* has first made possible the use of level 3 BLAS resulting in impressive speedups. Special data storages for the expansions either by *rows* or by *slices* have then enabled us to avoid the additional cost of *recopies*, especially for low precisions.

Precise operation counts and memory requirements have also been given for each scheme and for both $M2L$ kernel heights. This as well as CPU times from practical simulations have been used to precisely compare all these different versions. It appears that the BLAS version and the FFT improvement with blocks are the most efficient versions. While the BLAS version is always faster in case of double height kernel, the block FFT is faster with single height kernel for high precisions. However the memory requirements of the block FFT as well as the remaining numerical instabilities, precisely for high precisions, limit severely the benefit it offers for runtime in this case. This is reinforced when comparing the tradeoff between practical accuracy and runtime among all versions: the BLAS with double height kernel is then always the most efficient, while introducing no numerical instabilities and low extra memory requirements.

It has to be noted that in [GR97] and in [CGR99], a new version of the FMM has been introduced. Based on exponential or "plane wave" expansions, it leads to impressive speedup compared to the original formulation of the FMM. This new version requires however more complex mathematical background and thus more implementation work. Since no free code of this new version is currently available, we have not yet been able to compare it with our BLAS version. Nevertheless, it has to be noticed that this new version imposes tedious adaptation between the error introduced by the multipole and local expansions of the FMM and the error due to the use of plane waves: only a few precisions are thus available. The use of plane waves is yet promising and is still investigated (see [DH04]).

In the future, we plan to apply BLAS to the adaptive version of the FMM (see [CGR88] or [NKLW94]), and then to see how this will match its parallelization for both uniform and non uniform distributions on distributed memory architectures.

Acknowledgements

We are grateful to Guillaume Latu, ICPS-LSIIT (Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection) and INRIA CALVI project, for several helpful discussions on the underlying techniques of BLAS and their integration in the FMM.

A Appendix: Spherical harmonics

We introduce here the spherical harmonics which are the basis of the multipole and local expansions used in the FMM, and the associated Legendre functions they are themselves based on.

A.1 Associated Legendre functions.

In $3D$ space we use the following convention, commonly used in physics, for the spherical coordinates : θ denotes the co-latitudinal coordinate (with $\theta \in [0, \pi]$) and ϕ the longitudinal coordinate (with $\phi \in [0, 2\pi[$, ϕ corresponds to the polar angle in polar coordinates) as shown in figure A.1.

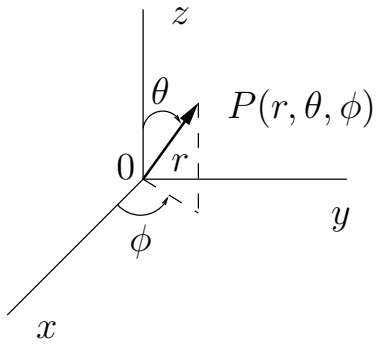


Figure A.1: Spherical coordinates of point $P(r, \theta, \phi)$.

We define the Legendre polynomials with Rodrigues' formula:

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} \left(x^2 - 1\right)^l \qquad \forall\, l \geq 0.$$

The associated Legendre functions, P_l^m, are defined by:

$$P_l^m(x) = (-1)^m (1 - x^2)^{m/2} \frac{d^m}{dx^m} P_l(x) \qquad \forall\, (l, m) \in \mathbb{N}^2 \quad with \quad 0 \leq m \leq l.$$

We extend the definition of those functions to negative values of m (with still $|m| \leq l$) as:

$$P_l^{-m}(x) = (-1)^m \frac{(l-m)!}{(l+m)!} P_l^m(x). \tag{A.1}$$

The following properties allow an efficient computation of those functions:

$$(l - m) P_l^m = x(2l - 1) P_{l-1}^m - (l + m - 1) P_{l-2}^m,$$

$$P_m^m = (-1)^m (2m - 1)!! (1 - x^2)^{m/2},$$

where $n!!$ denotes the product of all odd integers equal or lower than n. And:

$$P_{m+1}^m = x(2m + 1) P_m^m.$$

Here are the first associated Legendre functions:

$$\begin{array}{llll} P_0^0(x) = 1 & P_1^0(x) = x & P_2^0(x) = \frac{1}{2}(3x^2 - 1) & \dots \\ & P_1^1(x) = -(1 - x^2)^{\frac{1}{2}} & P_2^1(x) = -3x(1 - x^2)^{\frac{1}{2}} & P_3^3(x) = -15(1 - x^2)^{\frac{3}{2}} \\ & & P_2^2(x) = 3(1 - x^2) & \end{array} \qquad \begin{array}{l} \dots \\ P_4^3(x) = -105x(1 - x^2)^{\frac{3}{2}} \\ P_4^4(x) = 105(1 - x^2)^2 \\ \dots \end{array}$$

Finally, we have the following property of parity:

$$P_l^m(-x) = (-1)^{m+l} P_l^m(x).$$

A.2 Spherical harmonics.

With the associated Legendre functions P_l^m extended to negative values of m, as defined in (A.1), a common definition of the spherical harmonic of *degree* l and *order* m, with $l \geq 0$ et $-l \leq m \leq l$ is:

$$\Upsilon_l^m(\theta,\phi) = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta) e^{im\phi}.$$

However, as mentioned by [Gre88], the normalisation factor $\sqrt{\frac{(2l+1)}{4\pi}}$ can be discarded in the FMM computation. We therefore consider hereafter spherical harmonics defined without this normalisation factor. Moreover in order to simplify the formulae for the operators used in the FMM (see section 1.2.1), we introduce ϵ_m defined by:

$$\epsilon_m = \begin{cases} (-1)^m & \text{if } m \geq 0, \\ 1 & \text{otherwise,} \end{cases}$$

so that our (unnormalized) spherical harmonics are defined by:

$$Y_l^m(\theta,\phi) = \epsilon_m \sqrt{\frac{(l-m)!}{(l+m)!}} P_l^m(\cos\theta) e^{i.m.\phi}, \tag{A.2}$$

which can also be written (without the P_l^m with negative orders):

$$Y_l^m(\theta,\phi) = (-1)^m \sqrt{\frac{(l-|m|)!}{(l+|m|)!}} P_l^{|m|}(\cos\theta) e^{i.m.\phi}.$$

Remark A.1. *This definition corresponds to the one used by Epton & Dembart [ED95].*

When normalized, the spherical harmonics form an orthonormal basis for functions $f(\theta,\phi) \in \mathbb{R}$. Moreover we emphasize that our spherical harmonics are considered as identically null for $l < 0$ or $|m| > l$.

In particular we have:

$$Y_l^0(\theta,\phi) = P_l(\cos\theta),$$

and also the following property among opposite orders:

$$Y_l^{-m} = \overline{Y}_l^m \tag{A.3}$$

where $\overline{z}$ denotes the complex conjugate of $z \in \mathbb{C}$.

When considering a vector $\mathbf{Z}$ of spherical coordinates $\mathbf{Z} = (r,\theta,\phi)$ and its opposite vector $-\mathbf{Z}$ whose spherical coordinates are $-\mathbf{P} = (r, \pi-\theta, \pi+\phi)$, we have then:

$$Y_l^m(\pi-\theta,\pi+\phi) = (-1)^l Y_l^m(\theta,\phi).$$

At last we have the following well-known theorem (see [Gre88]):

Theorem 5 (Addition Theorem for spherical harmonics). *Given 2 vectors* $\mathbf{P_1} = (r1,\theta1,\phi1)$ *and* $\mathbf{P_2} = (r2,\theta2,\phi2)$ *and* γ *being the angle between the 2 vectors defined by:* $\cos\gamma = \cos\theta_1\cos\theta_2 + \sin\theta_1\sin\theta_2\cos(\phi_1-\phi_2)$, *the Addition Theorem for our spherical harmonics is:*

$$P_l(\cos\gamma) = \sum_{m=-l}^{l} Y_l^m(\theta_1,\phi_1) Y_l^{-m}(\theta_2,\phi_2).$$

B Appendix: Discrete Fourier Transform theory

These are classical definitions and results of Discrete Fourier Transform (see for example [PTVF92] or [BvH95]).

In most case, we implicitely consider that the period N is even, so that $\frac{N}{2}$ is an integer. It is straightforward to derive the corresponding formulae for odd N.

B.1 Convolution

Let $(f_l)_{l\in\mathbb{N}}$ and $(g_l)_{l\in\mathbb{N}}$, two *periodic* sequences with period N, we define the discrete convolution, $h = f * g$, of these two sequences by:

$$h_k = (f*g)_k = \sum_{l=-\frac{N}{2}+1}^{\frac{N}{2}} f_l g_{k-l} \qquad \forall\, k \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!]. \tag{B.1}$$

This can also be written as:

$$h_k = \sum_{\substack{m+l\equiv k[N] \\ (m,l)\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]^2}} f_l g_m \tag{B.2}$$

where $a \equiv b[N]$ means that a and b are "congruent modulo m".

A $2D$ convolution of 2 sequences $(f^l_n)_{(n,l)\in\mathbb{N}^2}$ and $(g^l_n)_{(n,l)\in\mathbb{N}^2}$, both periodic with respective periods J and K, is defined by:

$$h^k_j = (f \ * \ g)^k_j = \sum_{n=-\frac{J}{2}+1}^{\frac{J}{2}} \sum_{l=-\frac{K}{2}+1}^{\frac{K}{2}} f^l_n g^{k-l}_{j-n}.$$

B.2 DFT: Discrete Fourier Transform

Let $(f_l)_{l\in\mathbb{N}}$ denotes a periodic sequence of period N, we define the (forward) DFT of size N of f by:

$$\widehat{f}_k = \frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n \omega_N^{-nk} \qquad \forall\, k \in \mathbb{N}$$

where $\omega_N^j = e^{\frac{i2\pi j}{N}}$ (and $i = \sqrt{-1}$).

The sequence $\widehat{f}$ is also periodic with period N.

B.3 BDFT: Backward Discrete Fourier Transform

Let $(\widehat{f}_l)_{l\in\mathbb{N}}$ denotes a periodic sequence with period N, we define the BDFT of size N of $\widehat{f}$ by:

$$f_n = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \widehat{f}_k \omega_N^{nk} \qquad \forall\, n \in \mathbb{N}.$$

B.4 The theorem of Discrete Convolution

Theorem 6 (Discrete Convolution Theorem). *Let $(f_n)_{n\in\mathbb{N}}$ and $(g_n)_{n\in\mathbb{N}}$ be 2 periodic sequences with period N, we respectively denote $(\widehat{f}_k)_{k\in\mathbb{N}}$ and $(\widehat{g}_k)_{k\in\mathbb{N}}$ their DFT of size N. The DFT $\widehat{h}_k$ of their convolution h_k is then equal to:*

$$\widehat{h}_k = N\widehat{f}_k\widehat{g}_k.$$

A proof can be found in [BvH95]. The $\widehat{h}_k$ are periodic with period N.

B.5 The theorem of $2D$ Discrete Convolution

The application to $2D$ convolutions is straightforward.

Theorem 7 ($2D$ Discrete Convolution Theorem). *Let $(f_j^k)_{(j,k)\in\mathbb{N}^2}$ and $(g_j^k)_{(j,k)\in\mathbb{N}^2}$ be 2 sequences defined on $\mathcal{J}\times\mathcal{K}$ and extended with periodicity to $\mathbb{N}^2$.*

We note J and K the 2 periods. For example: $\mathcal{J}=[\![-\frac{J}{2}+1,\frac{J}{2}]\!]$ and $\mathcal{K}=[\![-\frac{K}{2}+1,\frac{K}{2}]\!]$, where J and K are even.

Their $2D$ (Forward) Discrete Fourier Transform of size $J\times K$ are defined by:

$$\hat{f}_j^k=\frac{1}{JK}\sum_{n\in\mathcal{J}}\sum_{l\in\mathcal{K}}f_n^l\omega_J^{-nj}\omega_K^{-lk}\qquad\forall\ (j,k)\in\mathcal{J}\times\mathcal{K},$$

and their $2D$ Backward DFT as:

$$f_j^k=\sum_{n\in\mathcal{J}}\sum_{l\in\mathcal{K}}\hat{f}_n^l\omega_J^{nj}\omega_K^{lk}\qquad\forall\ (j,k)\in\mathcal{J}\times\mathcal{K}.$$

We have then:

$$\hat{h}_j^k=JK\hat{f}_j^k\hat{g}_j^k\qquad\forall\ (j,k)\in\mathcal{J}\times\mathcal{K}.$$

B.6 Non-periodicity and zero-padding

A convolution of size N between two *non-periodic* sequences $(f_l)_{l\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]}$ and $(g_l)_{l\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]}$, can be defined by:

$$h_k=\sum_{\substack{l+m=k\\(l,m)\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]^2}}f_lg_m\qquad\forall\ k\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!].$$

We can relate this back to (B.2) by defining 2 sequences $(\widetilde{f}_l)_{l\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]}$ and $(\widetilde{g}_l)_{l\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]}$ as:

$$\forall\ l\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!],\qquad\widetilde{f}_l=\begin{cases}f_l & \text{if } l\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!],\\ 0 & \text{otherwise,}\end{cases}$$

and likewise for g_l.

We then extend the domain of these sequences to $\mathbb{N}$ thanks to M periodicity.

Now let:

$$\widetilde{h}_k=\sum_{\substack{m+l\equiv k[N]\\(m,l)\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]^2}}\widetilde{f}_l\widetilde{g}_m\qquad\forall\ k\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!].$$

Remark B.1. *The $\widetilde{h}_k$ with $k\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]\setminus[\![-\frac{N}{2}+1,\frac{N}{2}]\!]$ are not necessary null.*

By choosing M "big enough" we can have:

$$\widetilde{h}_k=h_k\qquad\forall\ k\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!].$$

When the convolution is "centered" (around 0, as in (B.2) for example), we usually need $M\geq\frac{3N}{2}$ in order to have: $\widetilde{h}_k=h_k$, $\forall\ k\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]$. But when the convolution is not "centered", as this is the case for $M2L$ (see below), we usually need to have $M\geq 2N$ (see [ED95] and [Ell95]).

This use of extra zeros is called *zero-padding*.

We can now use the Discrete Convolution Theorem (theorem 6) in order to compute the $\widetilde{h}_k$ and then the h_k (with backward DFT) for $k\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]$.

References

[And92] Christopher R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM Journal on Scientific and Statistical Computing*, 13(4):923–947, July 1992.

[AS72] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*. Tenth Printing, December 1972.

[BLA] Fortran 77 reference implementation source code of the blas from netlib. Electronically available at: `http://www.netlib.org/blas`.

[BvH95] William L. Briggs and Emden van Henson. *The DFT, An Owner's Manual for the Discrete Fourier Transform*. 1995.

[CBC+01] J. C. Carr, R. K. Beatson, J.B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *ACM SIGGRAPH 2001*, pages 67–76, Los Angeles, CA, August 2001.

[CGR88] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM Journal on Scientific and Statistical Computing*, 9(4):669–686, July 1988.

[CGR99] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155:468–498, 1999.

[CIGR99] Cheol Ho Choi, Joseph Ivanic, Mark S. Gordon, and Klaus Ruedenberg. Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion. *Journal of Chemical Physics*, 111:8825–8831, 1999.

[Dar99] Eric Darve. *Méthodes multipôles rapides : résolution des équations de Maxwell par formulations intégrales*. PhD thesis, Université Paris 6, 1999.

[DCHD90] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990.

[DCHH88] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1):1–17, March 1988.

[DDSvdV98] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers (Software, Environments, Tools)*. SIAM, Philadelphia, PA, USA, 1998.

[DH04] Eric Darve and Pascal Havé. Efficient fast multipole method for low-frequency scattering. *Journal of Computational Physics*, 197(1):341–363, June 2004.

[EB96] William D. Elliott and John A. Board, Jr. Fast Fourier Transform accelerated fast multipole algorithm. *SIAM Journal on Scientific Computing*, 17(2):398–415, 1996.

[ED95] Michael A. Epton and Benjamin Dembart. Multipole translation theory for the three-dimensional Laplace and Helmholtz equations. *SIAM Journal on Scientific Computing*, 16(4):865–897, 1995.

[Ell95] W.D. Elliott. Multipole algorithms for molecular dynamics simulation on high performance computers. Technical Report 95-003, Duke University, Department of Electrical Engineering, 1995. Doctoral dissertation.

[FJ05] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

[GD03] Nail A. Gumerov and Ramani Duraiswami. Recursions for the computation of multipole translation and rotation coefficients for the 3-D Helmholtz equation. *SIAM Journal on Scientific Computing*, 25(4):1344–1381, 2003.

[GR87] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[GR88] L. Greengard and V. Rokhlin. The rapid evaluation of potential fields in three dimensions, in vortex method. In C. Anderson and G. Greengard, editors, *Vortex Methods*, volume 1360 of *Lecture Notes in Mathematics*, pages 121–141. Springer-Verlag, 1988.

[GR97] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6:229–269, 1997.

[Gre88] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems.* MIT Press, Cambridge, MA, 1988.

[HJ96] Yu Hu and S. Lennart Johnsson. Implementing O(N) N-body algorithms efficiently in data-parallel languages. *Scientific Programming*, 5(4):337–364, 1996.

[KLvL98] Bo Kågström, Per Ling, and Charles van Loan. GEMM-based level 3 BLAS: high-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Softw.*, 24(3):268–302, 1998.

[LHKK79] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Transactions on Mathematical Software*, 5(3):308–323, September 1979.

[NKLW94] Keith Nabors, F. T. Korsmeyer, F. T. Leighton, and Jacob White. Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory. *SIAM Journal on Scientific Computing*, 15(3):713–735, May 1994.

[NW91] Keith Nabors and Jacob White. Fastcap: A multipole accelerated 3-D capacitance extraction program. *IEEE Transactions on Computer-Aided Design*, 10(11):1447–1459, November 1991.

[PSS95] H.G. Petersen, E.R. Smith, and D. Soelvason. Error estimates for the fast multipole method. II. The three-dimensional case. *Proceedings of the Royal Society of London. Series A, Mathematical and physical sciences*, 448:401–418, 1995.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing, Second Edition.* Cambridge University Press, 1992.

[Ran99] W.T. Rankin. Efficient parallel implementations of multipole based N-body algorithms. PhD. Dissertation, Duke University, Department of Electrical Engineering, April 1999.

[SHT+95] Jaswinder Pal Singh, Chris Holt, Takashi Totsuka, Anoop Gupta, and John Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-Hut, fast multipole, and radiosity. *Journal of Parallel and Distributed Computing*, 27:118–141, 1995.

[SP97] D. Soelvason and H.G. Petersen. Error estimates for the fast multipole method. *Journal of Statistical Physics*, 86:391–420, 1997.

[SP01] Xiaobai Sun and Nikos P. Pitsianis. A matrix version of the fast multipole method. *SIAM Review*, 43(2):289–300, 2001.

[WHG94] C. A. White and M. Head-Gordon. Derivation and efficient implementation of the fast multipole method. *Journal of Chemical Physics*, 101(8):6593–6605, October 1994.

[WHG96] Christopher A. White and Martin Head-Gordon. Rotating around the quartic angular momentum barrier in fast multipole method calculations. *Journal of Chemical Physics*, 105(12):5061–5067, 1996.

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399